НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:

Завідувач кафедри

_____ Олександр Коваль

«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційні технології моніторингу довкілля»

спеціальності 122 «Комп'ютерні науки та інформаційні технології»

на тему: «Система гарантування якості медичних додатків в розподіленому середовищі»

Виконав (-ла):
студент (-ка) IV курсу, групи ТМ-62
Сапон Ольга Миколаївна _____

Керівник:
старший викладач
Бандурка Олена Іванівна _____

Консультант:

_____

Рецензент:
доцент кафедри АЕС і ІТФ, к.т.н.
Баранюк Олександр Володимирович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 122 Комп’ютерні науки та інформаційні технології

Спеціалізація Інформаційні технології моніторингу довкілля

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту
Сапон Ольга Миколаївна
(прізвище, ім’я, по батькові)

1. Тема роботи “Система гарантування якості медичних додатків в розподіленому середовищі”
керівник роботи Бандурка Олена Іванівна, старший викладач
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”25” травня 2020р. № **1168-с**

2. Строк подання студентом роботи _____
3. Вихідні дані до роботи мова програмування Java, середовище розробки IntelliJ IDEA 2019, інструмент для автоматизації Appium

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати задачу та процес тестування, провести аналіз засобів розробки, створити алгоритм та розробити систему гарантування якості медичного додатку

5. Перелік ілюстративного матеріалу мета розробки, процес тестування, порівняння вартості AQA та MQA у часі, порівняння приладів тестування, засоби розробки, схема виконання тестів, робота програмної системи, статус виконання тестів, вікно Appium, висновки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання “ 11 ” жовтня 2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	16.09.19	
2.	Вивчення та аналіз задачі	1.10.19 – 29.11.19	
3.	Розробка архітектури та загальної структури системи	2.12.19 – 20.01.20	
4.	Розробка структур окремих підсистем	21.01.20 – 24.02.20	
5.	Програмна реалізація системи	25.02.20 – 30.03.20	
6.	Оформлення пояснювальної записки	31.03.20 – 29.05.20	
7.	Захист програмного продукту	09.06.20	
8.	Передзахист	09.06.20	
9.	Захист	18.06.20	

Студент _____ Ольга САПОН
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Олена БАНДУРКА
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Дипломну роботу розроблено на 57 аркушах, вона містить 13 рисунків, 6 таблиць та 23 бібліографічних найменувань за “Списком використаних джерел”. Також робота включає 4 додатки.

Метою даної дипломної роботи є опанування процесу тестування та створення системи гарантування якості мобільного додатку.

Для реалізації системи було обрано і проведено для нативного типу додатку автоматизоване тестування та використано емулятор, як прилад для тестування, що є зручним та зрозумілим для користувача.

Ключові слова: гарантування якості, процес тестування, медичний додаток, автоматизоване тестування, інструмент автоматизації.

ABSTRACT

The thesis is developed on 57 sheets, it contains 13 figures, 6 tables and 23 bibliographic titles according to the "List of used sources". The work also includes 4 applications.

The purpose of this thesis is to master the testing process and create a quality assurance system for the mobile application.

To implement the system, automated testing was selected and performed for the hybrid type of application and an emulator was used as a testing device that is convenient and user-friendly.

Keywords: quality assurance, testing process, medical application, automated testing, automation tool

ЗМІСТ

ВСТУП

Сучасна цивілізація досягла високого розвитку, але прогрес не стоїть на місці: покращуються та оновлюються вже існуючі технології, створюються нові, більш довершені та необхідні для полегшення та оптимізації умов життя людей.

Останнім часом більшість населення широко застосовує технологічні здобутки. До них належать і телефонні пристрої. Цей факт сприяє створенню великої кількості програмних продуктів, спрямованих для використання на операційних системах і платформах для мобільних телефонів та планшетних комп'ютерів Android та iOS.

На сьогоднішній день масивний об'єм операцій, які раніше виконувалися лише в певному закладі, людина має змогу обробити вдома, не витрачаючи на це багато часу. Це стосується різноманітних дій: починаючи з оплати товарів з інтернет-магазину і, закінчуючи використанням медичних додатків, для отримання направлення від лікаря на необхідну процедуру. У найближчому майбутньому людство планує перевести в режим мобільного користування велику кількість операцій, які є базовими та необхідними для комфортного проживання, розвитку та вдосконалення.

Зараз широкого поширення набувають медичні додатки.

Це пов'язане з тим, що вони зберігають час лікарів та пацієнтів; пацієнт має вільний доступ до перегляду результатів досліджень та аналізів, історії хвороби та внесення необхідних даних, які стосуються стану здоров'я; більш точно можна визначити діагноз, його особливості, отримати діаграми, аналізувати обрані показники стану, сформувати таблиці та способи лікування і підтримки чудового самопочуття пацієнта – і це все з будь-якої точки планети, бо немає потреби запам'ятовувати великі об'єми інформації та мати при собі дані про пацієнтів, у

друкованому вигляді. Саме тому додатки необхідно ретельно тестувати, щоб мати точні введені дані, результати аналізу та адекватну роботу програмного продукту.

Тестування необхідно тому, що всі ми робимо помилки. Деякі з них можуть бути незначними, в той час як інші – мати руйнівні наслідки. Все, що виробляється людиною, може містити помилки. Саме тому будь-який продукт потребує перевірки – тестування, перш ніж його можна буде ефективно і безпечно використовувати.

Аналогічні вимоги і для програмного забезпечення.

Програмне забезпечення – комп'ютерні програми, функції, а також супроводжуюча їх документація та дані, що мають відношення до експлуатації комп'ютерної системи.

Тестування програмного забезпечення – процес аналізу програмного засобу і супутньої документації з метою виявлення дефектів і підвищення якості продукту [1].

Тестування є один з найважливіших та ключових етапів розробки програмного продукту, особливо, якщо додаток відноситься до медичної сфери застосування і пов'язаний зі здоров'ям людини.

Пояснювальна записка містить п'ять розділів, кожен з них надає достатню інформацію для розуміння та усвідомлення створеної роботи та тестування загалом. У першому розділі описується задача гарантування якості. У другому розділі присутнє інформування аналізу процесу тестування. У третьому вказуються основні засоби розробки даної системи. У четвертому розділі знаходиться опис алгоритму створення та програмної реалізації. А у п'ятому – робота з програмної системою.

1. ЗАДАЧА ГАРАНТУВАННЯ ЯКОСТІ

Для виконання роботи необхідно опрацювати та усвідомити інформацію та літературні джерела, які стосуються саме гарантування якості продукту в цілому, дані про тестування:

- визначення;
- навіщо воно потрібне при розробці програмної системи;
- цілі;
- мета;
- типи тестування;
- розвінчати або підтвердити міфи, які звичайні люди та і часом розробники приймають за дійсність.

Найголовніше – це зрозуміти самому і донести до людей, наскільки цей процес є важливим під час розробки будь-якого програмного продукту або навіть проекту, який немає жодного відношення до ІТ-сфери.

Необхідно мати кваліфікованих працівників-тестерів, щоб мати можливість якісно виконувати розробку програмної система, тим самим задовольняти потреби замовників та користувачів. Тестер зі свого боку повинні цікавитися, дізнаватися та освоювати новітню інформацію і технології, бо в наш час всі процеси стрімко розвиваються, оновлюються та оптимізуються, особливо в сферах комп’ютерних технологій та розробки програмних забезпечень.

1.1. Загальна інформація про тестування

Тестування програмного забезпечення – це процес або серія процесів, призначених для того, щоб переконатися, що комп'ютерний код робить те, для чого він був розроблений, і, навпаки, що він не робить нічого ненавмисного. Тестування завжди присутнє в життєвому циклі розробки програмного продукту (рисунок 1). Програмне забезпечення повинно бути передбачуваним та послідовним, не представляючи сюрпризів для користувачів. Хоча завдання може бути непростим, адекватне тестування програми є дуже необхідною і досяжною частиною розробки програмного забезпечення.

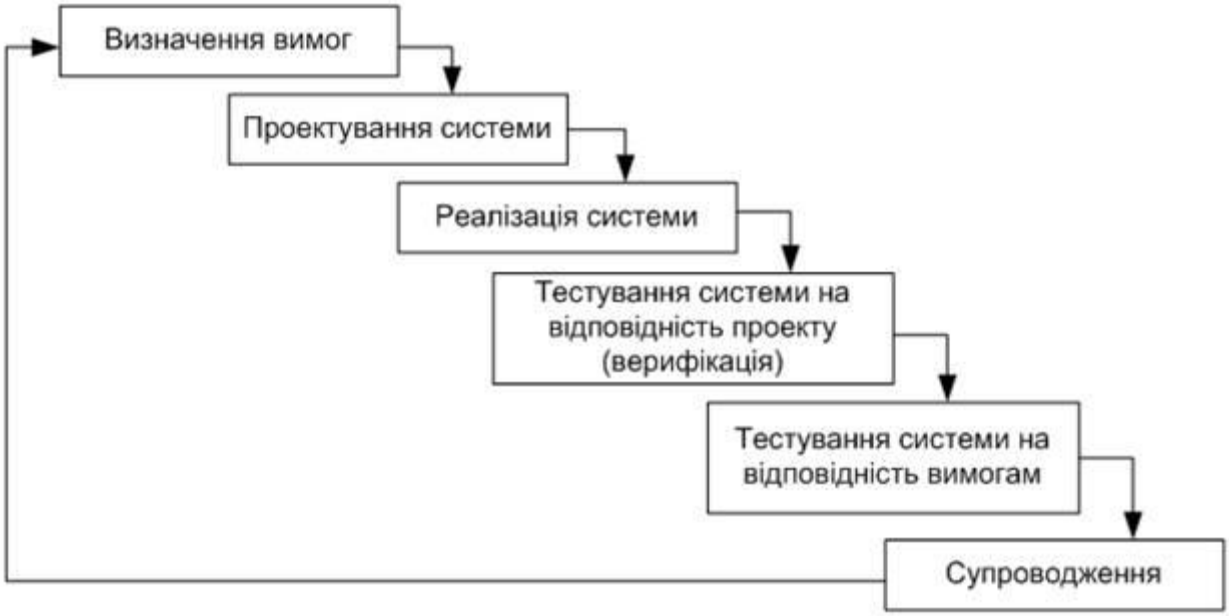

Рисунок 1 – Життєвий цикл розробки програмного продукту

Тестування програмного забезпечення – це технічне завдання, так, але воно також включає деякі важливі міркування економіки та психології людини.

В ідеальному світі ми хотіли б перевірити всі можливі перестановки програми. Однак у більшості випадків це просто неможливо. Навіть на перший погляд проста

програма може мати сотні чи тисячі можливих комбінацій вводу та виводу. Створювати тестові приклади для всіх цих можливостей недоцільно. Повне тестування складної програми зайняло б занадто багато часу і вимагало б занадто багато людських ресурсів, щоб бути економічно можливим [2].

1.2. Мета та ціль гарантування якості

Однією з головних причин поганого тестування додатків є той факт, що більшість програмістів починають з помилкового визначення терміну. Вони хибно вважають, що :

- тестування – це процес демонстрації того, що помилок немає;
- мета тестування – показати, що програма виконує призначені функції правильно;
- тестування – це процес встановлення впевненості в тому, що програма робить те, що має робити.

Ці визначення перевернуті.

Під час тестування програми на меті є додавання їй певної цінності. Додавання значення через тестування означає підвищення якості або надійності програми. Підвищення надійності програми означає пошук та усунення помилок. Тому не тестуйте програму, щоб довести, що вона працює; швидше, почніть з припущення, що програма містить помилки (дійсне припущення майже для будь-якої програми), а потім протестуйте програму, щоб знайти якомога більше помилок. Таким чином, більш прийнятним є таке визначення: Тестування – це процес виконання програми з наміром знайти помилки.

Більшість керівників проектів посилаються на тестовий випадок, який не виявив помилку “успішного тестового запуску”, тоді як тест, який виявляє нову помилку, зазвичай називають “невдалим”. Знову це стає перевернуто. “Невдале” позначає щось небажане або невтішне. На наш спосіб мислення, добре побудований

і виконаний тест програмного забезпечення є успішним, коли він виявляє помилки, які можна виправити. Тестовий випадок, який виявив нову помилку, навряд чи можна вважати невдалим [2].

Цілі тестування можуть відрізнятися, в залежності від етапу розробки програмного забезпечення, на якому воно проводиться. Наприклад, на етапі кодування метою тестування буде виклик як можна більшої кількості збоїв в роботі програми, що дозволить локалізувати та виправити дефекти. Водночас, при приймальному тестуванні необхідно показати, що система працює правильно. У період супроводу, тестування в основному необхідно для того, щоб переконається у відсутності нових багів, що з'явилися під час внесення змін. Головне ж завдання тестування – пошук дефектів.

1.3. Вхідні, вихідні дані та порядок виконання

Введення зазвичай вважаються даними, введеними на клавіатурі. Хоча це є важливим джерелом введення системи, дані можуть надходити з інших джерел – дані з систем взаємодії, дані з пристроїв взаємодії, дані, зчитувані з файлів або баз даних, стан, у якому знаходиться система коли дані надходять, і середовище, в якому система виконує [3].

Вихідні дані мають таку саму різноманітність. Часто виходи розглядаються як просто дані, що відображаються на екрані комп'ютера. Крім того, дані можуть надсилатися в системи взаємодії і на зовнішні пристрої. Дані можна записувати у файли чи бази даних. Стан або середовище можуть бути змінені виконанням системи. Усі ці вхідні та вихідні дані є важливими компонентами тестового випадку [3].

Існує два стилі проектування тестових випадків щодо порядку виконання тесту.

Каскадні тестові приклади – тестові випадки можуть будуватися один на одному. Наприклад, перший тестовий випадок виконує певну особливість програмного забезпечення, а потім залишає систему в такому стані, що другий

тестовий випадок може бути виконаний. Кожен з цих тестів може бути побудований на попередніх тестах. Перевага полягає в тому, що кожен тестовий випадок, як правило, менший і простіший. Недоліком є те, що якщо один тест не вдасться, наступні тести можуть бути недійсними.

Незалежні тестові приклади – кожен тестовий випадок є повністю самостійним. Тести не спираються на один одного і не вимагають успішного виконання інших тестів. Перевага полягає в тому, що будь-яка кількість тестів може бути виконана в будь-якому порядку. Недоліком є те, що кожен тест має тенденцію бути більшим і складнішим, а отже, складніше проектувати, створювати та підтримувати [3].

1.4. Деталізація теми

Тема даної роботи передбачає, перш за все, дослідження питання гарантування якості в сфері комп'ютерних здобутків, ознайомлення з програмних продуктом компанії, а саме медичним додатком, для передачі показників від датчиків, пов'язаних з сердечним ритмом, переглядом історії за необхідний проміжок часу, якщо ви зареєстровані як пацієнт, якщо ж як лікар, то перегляду кардіограми, загальних та детальних даних про своїх пацієнтів, таблиць та звітів про їх здоров'я. Також лікарі радять, основуючись результатами пацієнтів, ввести певні корективи під час фізичного навантаження та в стані спокою. У їх функціоналі присутня можливість створення певних груп для поділу своїх клієнтів та більш швидкій реакції на можливі зміни, бо цей додаток робить значний акцент для пацієнтів без діабету або з різним його ступенем. Це є досить розумно, бо таке умовне розмежування створює можливості як пацієнтам в перегляді рекомендацій та певних деталей стосовно свого стану, так і лікарям в створенні груп та додаванні до них людей різних категорій, що поліпшує та спрощує відстеження результатів їх замірів та створення аналізу на основі існуючих даних.

Метою створеної роботи є розробка системи, яка буде гарантувати якість мобільного додатку, вірне виконання всіх вищезазначених функцій та багатьох інших, що є необхідним перш за все для майбутніх користувачів. Також тестування грає важливу роль перед кожним оновленням додатку, тобто перед додаванням нового функціоналу у вже існуючий або редагуванням старого, що робить ще більш потрібним та актуальним створену систему тестування.

1.5. Висновки до розділу

В даному розділі було розглянуто загальну інформацію, для розуміння терміну, процесу та необхідності тестування.

Проведено аналіз його головної мети, бо часто приймають іншу ціль, не досить вірну. Також ознайомлення з вхідними і вихідними даними та стилі проектування тестових випадків щодо порядку виконання тесту.

Представлене детальне пояснення виконуваної роботи.

2. АНАЛІЗ ПРОЦЕСУ ТЕСТУВАННЯ

Для будь-якої системи різного розміру неможливо перевірити всі різні логічні шляхи та всі різні комбінації вхідних даних. З нескінченної кількості варіантів, кожен з яких гідний певного рівня тестування, тестери можуть вибрати лише дуже малу підмножину через обмеження ресурсів.

Тестування має бути невід’ємною частиною розробки програмного забезпечення. Є багато факторів, які слід враховувати при виборі стратегії тестування. Важливо вибирати тестові справи розумно. Відсутність дефекту може призвести до значних втрат для вашої організації, якщо несправна система буде розміщена у виробництві.

2.1. Ручне та автоматизоване тестування

Тестування програмного забезпечення є невід'ємною частиною розвитку успішного програмного проекту. Окрім ряду форматів тестування, існує також дуже активна дискусія про те, що автоматичне тестування краще, ніж тестування вручну.

Під час ручного тестування інженер-випробувач (людина) вручну виконує тестові справи. Це означає, що інженер-тестування описує кілька сценаріїв та кращих випадків, за якими вони хочуть перевірити функціонування. Ручне тестування не використовує жодних інструментів чи сценаріїв. Це означає, що інженеру-тестувальнику необхідно підготувати набір даних та сценарій та запустити правильні входи чи дії для тестування описаного сценарію.

За допомогою автоматизованого тестування все це – як випливає з назви – автоматизоване. Це означає використовувати сценарії та інструменти, які готують

дані та стан, а потім виконувати дії, необхідні для перевірки сценарію в автоматизованому вигляді.

Як ручне, так і автоматизоване тестування мають переваги та недоліки. Варто знати різницю та коли використовувати те чи інше для найкращих результатів.

Переваги ручного тестування

Під час ручного тестування інженер-тестувальник має повний контроль над виконанням кожної дії. Це зумовлює більше візуального зворотного зв'язку під час процесу, що дозволяє інженеру-тестувальнику налагоджувати або легше знаходити проблеми. Крім того, оскільки ручне тестування не вимагає ніяких інструментів, компанія витрачає менше коштів на тестування. Не варто забувати про витрати на наймання більшого персоналу для перевірки, щоб перевірити, чи працює ваш програмний продукт. Для невеликих змін або баз коду є більш доцільним швидко перевірити функцію, вручну протестувавши її, замість того, щоб налаштувати і розкрутити цілий набір тестування. Нарешті, ручне тестування добре працює для пошуку візуальних помилок та перевірки зручності використання програми. Ця частина тестування передбачає спостереження людини, щоб знайти будь-які збої – те, що неможливо (належним чином) виявити за допомогою автоматизованих інструментів.

Недоліки ручного тестування

Зазвичай для ручного тестування потрібно більше часу для завершення. Інженер тестування повинен підготувати набори даних, підготувати стан програми та виконати всі кроки для перевірки для завершення сценарію. Процес повільний і схильний до людських помилок. Всього одна помилка може означати, що інженер-тестувальник повинен робити все налаштування заново. Багато компаній вважають, що ручне тестування дешевше, оскільки їм не потрібно витрачати кошти на інструменти автоматизації тестів або на інструменти постійної інтеграції. Однак ці інструменти дешеві в порівнянні з пошуком та використанням декількох інженерів-тестувальників, які насправді виконують ту саму роботу, що й інструменти автоматизації. Багато тестів важко змоделювати за допомогою ручного тестування. Стресові тести – яскравий приклад цього. Стрес-тестування тестує програмне

забезпечення під великим навантаженням. Інженери-тестувальники використовують цей тип тестування, щоб знайти крапку програми та відчути, як вона поводиться під цим великим навантаженням. Часто стрес-тестування вимагає створення багатьох сотень чи тисяч запитів за короткий проміжок часу. Іноді для цього потрібно використовувати купу користувачів, які в той же момент підключаються до програми. Інженеру-тестувачу неможливо відтворити такий тип поведінки вручну.

Нарешті, складних сценаріїв часто уникають під час ручного тестування або перевіряються лише епізодично. Це залишає тестові прогалини у програмі. Автоматизація – це чудовий варіант для інженера-тестувальника для моделювання цих складних випадків.

Вище зазначене підтверджує, що ручне тестування має більше недоліків, ніж переваг, хоча воно все-таки має свої можливості. Основна помилка при ручному тестуванні полягає в тому, що лідери IT вважають, що вони економить гроші, не використовуючи інструменти платної автоматизації. Але вони забувають про приховані витрати на персонал більшої групи тестування, щоб заповнити розрив між ручним та автоматизованим тестуванням.

Переваги автоматизованого тестування

Автоматизоване тестування є надійним і завжди повертає той самий результат (відтворюваний). Оскільки кожен крок автоматизований, він не піддається людським помилкам, як ручне тестування.

Компаніям логічно інвестувати в інструменти автоматизації тестів, які допомагають їм налаштувати тестовий набір. Це дає їм змогу виконувати такі дії, як стрес-тестування, оскільки набір може імітувати тисячі клієнтів, які підключаються в один і той же момент.

Далі автоматизація тестів дозволяє запускати всі тести набагато швидше, ніж тестування вручну. Ви можете швидко імітувати складні сценарії.

Недоліки автоматизованого тестування

Автоматичне тестування не можна використовувати для виявлення візуальних помилок або помилок UX. Пошук помилок UX потребує людського ока. Тож автоматизовані інструменти тестування не покриють усі сценарії тестування.

Розробники можуть втратити цінний час налагодження неправильних сценаріїв тестування.

Автоматизоване тестування надійніше та швидше, ніж тестування вручну. Це підвищує продуктивність команди розробників; однак команда може витратити багато часу на тести налагодження.

Детальна інформація про різницю та схожість цих двох видів тестування зображена в таблиці 2.1.

Таблиця 2.1 – Порівняння видів тестування

Критерії	Ручне тестування	Автоматизоване тестування
Точність	Ручне тестування показує меншу точність через більші можливості людських помилок.	Тестування автоматизації показує більш високу точність завдяки комп'ютерному тестуванню, що виключає шанси помилок.
Тестування в масштабі	Ручне тестування потребує часу, коли тестування потрібно у великих масштабах.	Автоматичне тестування легко проводить тестування у великих масштабах з максимальною ефективністю.
Час обороту	Ручне тестування потребує більше часу, щоб пройти цикл тестування, і, таким чином, час обороту більше.	Автоматичне тестування завершує цикл тестування протягом рекордного часу, і, таким чином, час виконання значно нижчий.
Ефективність витрат	Ручне тестування потребує більших витрат, оскільки воно передбачає наймання експертів-експертів для проведення тестування.	Автоматичне тестування економить витрати, що виникають, оскільки інтегрована інфраструктура програмного забезпечення працює тривалий час.
Досвід користувача	Тестування вручну забезпечує кінцевому користувачеві програмного забезпечення високий досвід користувачів, оскільки воно вимагає спостереження і пізнавальних можливостей людини.	Тестування автоматизації не може гарантувати хорошого досвіду користувача, оскільки машині не вистачає спостережливості та когнітивних здібностей людини.

Продовження таблиці 2.1

Напрями спеціалізації	Для проведення кращих результатів слід використовувати ручне тестування для проведення дослідницьких, тестових зручностей та спеціальних тестів.	Для найкращих результатів слід використовувати автоматичне тестування для регресії, тестування навантаження, тестування продуктивності та повторного виконання.
Навички користувачів	Користувачі повинні мати можливість імітувати поведінку користувачів та будувати тестові плани для покриття всіх сценаріїв.	Користувачі повинні бути висококваліфікованими у програмуванні та розробці сценаріїв, щоб створити тестові випадки та автоматизувати якомога більше сценаріїв.

Використання ручного тестування

Ручне тестування підходить для тестування на юзабіліті, спеціальних тестів та дослідницьких тестів. Тестування юзабіліті зосереджено на вимірюванні зручності використання програми. Спеціальне тестування використовує вільний підхід, при якому інженер тестування намагається зламати компоненти без заданого сценарію. Останній підхід, дослідницьке тестування, зосереджується на знаннях, досвіді, аналітичних навичках тестера, творчості та інтуїції. Тут тест характеризується погано написаною специфікацією документації або коротким часом для виконання.

Використання автоматичного тестування

Автоматизоване тестування добре для регресійного тестування, перевірки працездатності, тестування навантаження та високо повторюваних функціональних тестів. Коротше кажучи, ви повинні автоматизувати кожен тест, який зможете.

2.2. Тестування мобільних додатків

Тестування для мобільних пристроїв абсолютно відрізняється від тестування програмних додатків, таких як веб- або настільні програми. У мобільних додатках

фізичні пристрої мають набагато більший вплив на програмне забезпечення, яке працює на них, порівняно з іншим програмним забезпеченням, таким як веб-програми. Оскільки на ринку існує стільки різних смартфонів, мобільним тестерам потрібно зосередитися на апаратному забезпеченні в процесі тестування. Крім того, користувачі, що рухаються навколо та використовують різні мережі передачі даних, змушують мобільних тестерів бути в русі під час тестування. Окрім обладнання, очікування користувачів відіграють важливу роль у повсякденному бізнесі мобільного тестера, і його потрібно поставити серйозно [6].

2.2.1. Головне мета під час тестування додатків

Користувач програми – головна спеціальність і головне завдання для мобільних команд. Той факт, що кожен користувач має унікальні очікування, ускладнює розробку та доставку “правильного” додатку для клієнтів. Як показують декілька звітів та опитувань, користувачі мобільних пристроїв мають набагато більші очікування щодо мобільних додатків порівняно з іншими програмними засобами, такими як браузерні програми. Більшість звітів та опитувань стверджують, що майже 80% користувачів видаляють додаток після першого використання! Перші чотири причини видалення завжди погані дизайн та зручність використання, час завантаження та збої відразу після встановлення. Майже 60% користувачів змушені видаляти додаток, який потребує реєстрації, тоді як більше половини користувачів очікують, що додаток запуститься за дві секунди. На основі цих номерів можна зробити висновок, що користувачі мобільних телефонів мають дуже високі очікування щодо використання, ефективності та надійності. Дуже багато додатків виконують одне і те ж завдання, це означає, що завжди існує принаймні один конкурентний додаток, що дозволяє користувачам дуже просто завантажувати інший додаток, оскільки це лише один дотик [6]. Основні моменти, яких слід дотримуватись, розробляючи та тестуючи мобільний додаток:

– зібрати інформацію про вашу можливу цільову групу;
– запитати своїх клієнтів про їхні потреби;
– ваш додаток повинен вирішити проблему для користувача;
– корисність справді важлива;
– додаток має бути надійним та надійним;
– продуктивність додатків дійсно важлива;
– програми повинні бути красивими.

2.2.2. Типи додатків

Найважливішим фактором, який слід враховувати під час тестового планування, є перевірка типу мобільних додатків. Ви головним чином натрапите на три типи мобільних додатків: мобільні веб-додатки, нативні додатки та гібридні (рисунок 2.1).

Рисунок 2.1 – Типи мобільних додатків

Мобільні веб-додатки (Mobile Web)

Веб-додатки – це не реальні програми; це насправді веб-сайти, які відкриваються у вашому смартфоні за допомогою веб-браузера. Мобільні веб-сайти мають найширшу аудиторію з усіх основних типів додатків.

Приклад: http://www.tutorialspoint.com/

Переваги:

- легкий доступ;
- легкий розвиток: Розробка чуйного дизайну та реструктуризація вмісту, який належним чином відображатиметься на меншому екрані / апаратному забезпеченні, зробить будь-який веб-сайт настільних пристроїв мобільним;
- легке оновлення: просто оновіть в одному місці, і всі користувачі автоматично отримають доступ до останньої версії сайту;
- не потрібно встановлення, порівняно з нативним або гібридним додатком.

Недоліки:

- мобільні веб-сайти не можуть використовувати деякі функції. Наприклад, доступ до файлової системи та місцевих ресурсів недоступний на веб-сайтах;
- багато існуючих веб-сайтів не підтримують офлайн-можливості;
- користувачі не матимуть значок програми на своєму головному екрані як постійне нагадування. Веб-сайт потрібно відкривати лише у веб-браузері;
- у той час як нативні та гібридні програми з’являються у магазині додатків та Google Play, веб-додатки не будуть. Тож перерозподіл не є таким розумним.

Нативні додатки(Native Apps)

Нативний додаток розроблено спеціально для однієї платформи. Його можна встановити через магазин додатків (наприклад, Google Play Store або Apple Store App).

Приклад: Whatsapp, Facebook.

Переваги:

- нативні додатки живуть на пристрої та отримують доступ до них через піктограми на головному екрані пристрою;

- вони можуть повністю скористатися всіма можливостями пристрою – вони можуть використовувати камеру, GPS, акселерометр, компас, список контактів тощо;
- нативні програми можуть використовувати систему сповіщень пристрою та працювати в автономному режимі;
- видавці можуть використовувати push-сповіщення, попереджуючи користувачів щоразу, коли публікується новий фрагмент вмісту або коли потрібна їхня увага;
- нативні додатки підтримують дизайн інтерфейсу кожної операційної системи, таким чином вони пропонують найкращий досвід користувача. Наприклад, вони можуть мати лівий вирівняний заголовок в Android та центральний заголовок в iOS.

Недоліки:

- висока вартість створення програми: Нативні програми, розроблені для однієї платформи, не працюватимуть на іншій платформі. Додаток, створений для Android, не працюватиме на iOS. Потрібно створити інший додаток для iOS. Через цю причину потрібно підтримувати кілька версій програми;
- незважаючи на те, що ви можете публікувати вбудовані програми, вам потрібно буде підтримувати веб-сайт для мобільних пристроїв, оскільки мобільний додає більше трафіку. Тож технічне обслуговування вище.

Гібридний додаток

Гібридні програми – це спосіб викрити вміст із існуючих веб-сайтів у форматі додатків. Їх можна добре описати як суміш Web App і Native App.

Приклад: Instagram, Wikipedia.

Переваги:

- розробка гібридного додатка дешевше, ніж розробка нативного. Він може бути побудований для крос-платформ, тобто знижена вартість на розробку додатків;
- технічне обслуговування просте, оскільки не так багато версій, які потрібно підтримувати;

– він може скористатися кількома функціями, наявними в пристрої;
– його можна знайти в App Store, що робить розповсюдження простим;
– він має вбудований браузер лише в програму.

Недоліки:

– графіка менш звикла до операційної системи порівняно з нативними;
– гібридні додатки повільніше, ніж нативні [7].

2.2.3. Прилад для тестування

Для проведення мобільного тестування потрібен мобільний пристрій. Це доступ до того, як працюватиме наш продукт і виглядатиме на певному мобільному наборі.

Припустимо, ми розробляємо заявку на систему бронювання авіаквитків. Після того, як продукт буде повністю розроблений, в рамках тестування мобільних пристроїв нам потрібно перевірити, чи працює програма, як очікується, з усіма використовуваними пристроями, такими як телефони Android, iOS, телефони Blackberry та інші різні типи планшетів та iPad.

Для проведення цього виду перевірки нам потрібно придбати кожен такий пристрій, і тоді ми можемо перевірити, чи програма веде себе відповідно до сподівань. Так, ви думали правильно, як власник продукту, заперечно, це буде дуже дорого придбати таку велику кількість мобільних пристроїв та провести тестування. Так чи є доступний розумний альтернативний варіант?

Рішення цієї проблеми полягає у використанні мобільних симуляторів та мобільних емуляторів. Це в першу чергу програмні програми, призначені для моделювання важливих функцій смартфона. Вони дуже схожі за своєю природою, тому іноді їх використовують взаємозамінно. Чим тестування на Емуляторі / Симуляторі відрізняється від тестування на реальному пристрої розібрано в таблиці 2.2.

Таблиця 2.2 – Порівняння приладів тестування

Критерії	Реальний прилад	Емулятор / Симулятор
Ціна	Отримання реальних пристроїв обійдеться вам дорожче.	Це майже безкоштовно, нам просто потрібно завантажити та встановити їх.
Швидкість обробки	Він має більш швидку обробку; проте затримка в мережі може бути нормальною.	Це повільніше порівняно з фактичними пристроями. Він спостерігав меншу затримку, ніж реальні пристрої, підключені до локальної мережі або в хмарі.
Налагодження	Налагодження не так просто.	Він забезпечує поетапну налагодження програми. Також він забезпечує ефективний спосіб зйомки скріншотів.
Тестування веб-додатків	Веб-додатки можна перевірити звичайним чином.	Тестувати веб-додаток набагато простіше.
Надійність	Тестування на реальному пристрої має головну перевагу в тому, що воно завжди дає точні результати.	Він не може імітувати всі типи взаємодій користувачів; отже, іноді це може призвести до помилкових результатів. Тож він набирає низькі показники щодо надійності.

Симулятор / емулятор не може імітувати наступні функції:

- акумулятор мобільного пристрою;
- камера мобільного пристрою;
- складно імітувати перебої, такі як вхідні дзвінки та SMS;
- не стільки реалістичне моделювання для використання пам'яті мобільних пристроїв.

Давайте тепер розберемося докладніше про мобільні симулятори та мобільні емулятори. Між ними є специфічні відмінності. У таблиці 2.3 перерахована основна різниця між симулятором та емулятором.

Таблиця 2.3 – Порівняння симулятора та емулятора

Критерії	Емулятор	Симулятор
Що це імітує	Програмне забезпечення для мобільних пристроїв. Обладнання для мобільних пристроїв. Мобільна операційна система.	Внутрішня поведінка пристрою. Це не імітує обладнання.
Як це отримати	Зазвичай надається виробником пристрою.	Зазвичай надається виробником пристрою або іншою компанією.
Внутрішня структура	Він написаний мовою монтажу на рівні машин.	Він написаний мовою високого рівня.
Налагодження	Він більше підходить для налагодження.	Він не підходить для налагодження.
Продуктивність	Емулятори дійсно повільні. Емуляція власне апаратного забезпечення зазвичай запускає програмне забезпечення повільніше, ніж раніше.	Швидше, ніж емулятори.
Приклад	Google SDK для Android	Apple’s симулятор iOS

Отже, який найкращий вибір для мобільного тестування? Найкраща практика свідчить про те, що, поки триває фактична розробка, ми повинні використовувати емулятор або тренажер. Перед тим, як доопрацювати продукт, слід провести перевірку стану безпеки з обраними реальними пристроями [7].

2.3. Інформація про мобільний додаток

Додаток Кардіолайз пропонує цілісний підхід для запобігання серцевих розладів для груп підвищеного ризику, використовуючи інноваційні та клінічно перевірені алгоритми машинного навчання, щодо віддаленого моніторингу здоров’я

серця в реальному часі, що забезпечує прості персоналізовані звіти, виявлення та прогноз на основі даних про небезпечні події серця.

Віддалений моніторинг здоров’я серця в режимі реального часу дозволяє точно визначити положення кожного серцевого параметра за шкалою від особистої базової лінії до патології для запобігання небезпечних подій серця для груп підвищеного ризику. Профілактика серцево-судинних захворювань на цукровий діабет – це комплексне рішення для первинної профілактики серцево-судинних ускладнень при діабеті з моніторингом конкретних серцевих параметрів, своєчасним виявленням та вилікуванням ризику серцево-судинні захворювання. Управління ризиками втоми – комплексне рішення щодо страхування відповідальності та система оповіщення для безперебійного моніторингу ризику втоми в реальному часі у виконавців критичної безпеки з метою запобігання подальших аварій.

2.4. Виділення обраних методів

Існує багато факторів, які слід враховувати при виборі стратегії тестування. Основне правило для тестування: впровадити якомога більше автоматизованого тестування. Тестування вручну відбувається повільно і не вписується в сучасні спритні практики. Переваги автоматизованого тестування набагато більше, ніж користі від ручного тестування. Графік залежності вкладених коштів в методи від витраченого часу тестування зображено на рисунку 2.2.

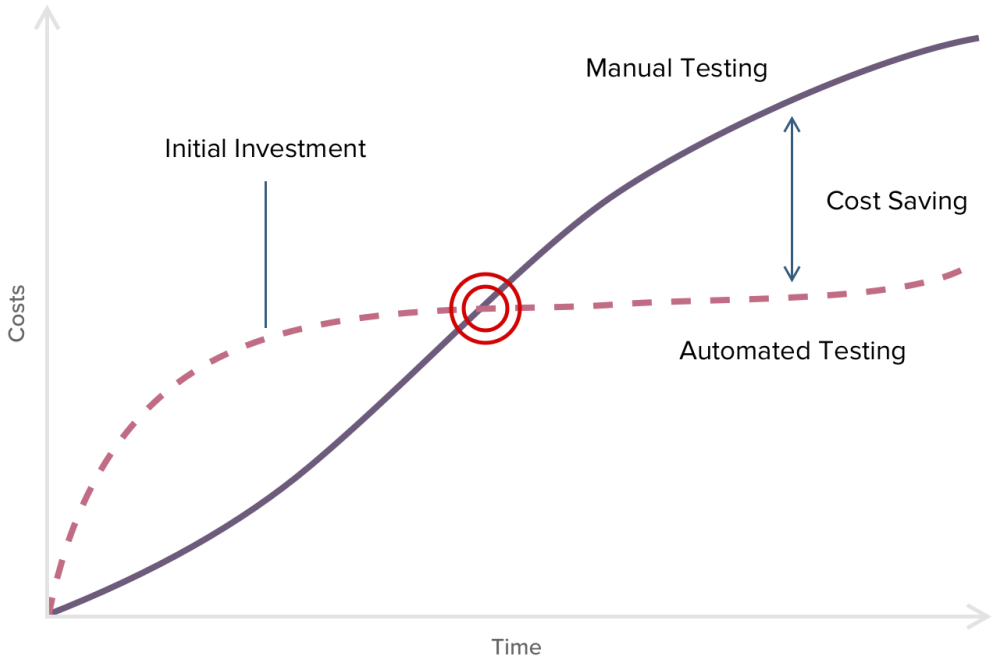

Рисунок 2.2 – Графік залежності вкладених коштів від часу

Для реалізації та виконання роботи обрано саме автоматизований процес тестування мобільного додатку. Вид обраного застосунку є нативний додаток. Більше інформації про ці різновиди можна дізнатися вище в розділі. Приладом для тестування обрано емулятор, тому що він – безкоштовний та завдяки цьому є можливість виконати тестування на пристроях з різними характеристиками.

2.5. Висновки до розділу

В даному розділі було розглянуто інформацію про ручне та автоматизоване тестування, а саме що кожен з них означає, переваги та недоліки, відмінності між ними та моменти для кращого їх використання.

Проведено аналіз мобільних додатків. Ознайомлення з головною метою під час їх тестування, інформування відносно видів додатків та приладів для тестування.

Представлене виділення обраних методів.

3. ЗАСОБИ РОЗРОБКИ

Під час тестування програмних продуктів необхідно уважно обирати засоби розробки, які б покращили та полегшили виконання запланованої роботи, використовуючи всі потрібні знаряддя для здійснювання поставленої мети.

Середовищем розробки системи було обрано IntelliJ IDEA 2019, інструментом автоматизації Appium та засобу автоматизації роботи з програмними проектами Apache Maven. Для розробки тестів була використана мова програмування Java.

3.1. Середовище розробки IntelliJ IDEA

IntelliJ IDEA – один з найбільш інноваційних та інтелектуальних середовищ розробки інтегрованих Java (IDE), доступних сьогодні.

IDEA – це повнофункціональний IDE, який розвинувся за межі традиційних IDE(інтегроване середовище розробки) за рахунок інтеграції сучасних функцій найкращої практики, таких як рефакторинг, інтелектуальна допомога в редагуванні коду та аналіз часу виконання коду. На стороні клієнта IDEA пропонує користувальницький інтерфейс дизайнера GUI(графічний інтерфейс користувача) для швидкого та нескладного дизайну інтерфейсу; на стороні сервера IDEA користується інтеграцією з технологіями J2EE(обчислювальна корпоративна платформа Java), включаючи провідні сервери додатків. Це єдиний IDE в галузі, який спочатку був побудований з метою підвищення продуктивності своїх користувачів, забезпечуючи при цьому зручне, налаштоване середовище розробки під індивідуальні смаки та стилі кодування.

IntelliJ IDEA забезпечує потужне середовище для управління проектами Java та вихідними файлами, а також створення та тестування програм. Підтримка популярних галузевих стандартів, таких як J2EE, Ant (java-утиліта для автоматизації процесу збирання програмного продукту), JUnit (бібліотека для тестування програмного забезпечення для мови Java), XML (розширювана мова розмітки) та різних систем управління ревізією, поєднує всі основні аспекти розвитку Java під одним інструментом.

Крім інтегрованих інструментів, вигадливого редактора та неясних функцій управління проектами, IDEA надає величезну кількість функцій, спрямованих на підвищення продуктивності та поліпшення досвіду розробки. IDEA може допомогти підвищити ефективність, усуваючи приземлені та повторювані завдання кодування, попереджаючи про це помилки перед компіляцією, відстеження змін коду та включення потужного рефакторингу. Заглибившись углиб IDEA та застосувавши ці функції, ви можете стати більш ефективним розробником Java, оскільки ви можете витрачати час на розробку проекту, а не на управління кодом.

Найвизначнішою особливістю IDEA є редактор вихідного коду. Редактор IDEA може автоматично відформатувати ваш вихідний код, перевірити його на наявність синтаксичних помилок і кольорових ключових слів Java – і все це в режимі реального часу під час введення тексту. Поінформованість редактора про структуру коду дозволяє переходити за методом або блокувати, згортати JavaDoc, ховати внутрішні заняття та скористатися іншими зручностями. Ви можете шукати та замінювати текст та навіть використання коду для всього проекту. У порівнянні з інструментами, такими як звичайний текстовий редактор, це різниця між пилочкою для нігтів та швейцарським армійським ножем. Необхідно багато інструментів, щоб допомогти розробнику створити додатки. Сучасні IDE прагнуть об'єднати всі ці інструменти в згуртовану систему, роблячи доступ до інструментів більш простим та продуктивним [9].

3.2. Мова програмування Java

Java – мова програмування загального призначення. Вона має функції підтримки програмування на основі об'єктно-орієнтованих, процедурних та функціональних парадигм.

Мова програмування не орієнтована на об’єкти. Саме парадигма є об'єктно-орієнтованою, а мова програмування може мати функції, які спрощують реалізацію об’єктно-орієнтованої парадигми. Іноді програмісти мають помилкові уявлення, що всі програми, написані на Java, завжди орієнтовані на об’єкти. Java також має функції, що підтримують процедурні та функціональні парадигми.

Початкова версія платформи Java була випущена Sun Microsystems (частина корпорації Oracle з січня 2010 року) у 1995 році. Розробка мови програмування Java була розпочата в 1991 році. Спочатку мова називалася Oak, і її мали використовувати в телевізори для телевізорів.

Незабаром після виходу Java стала дуже популярною мовою програмування. Однією з найважливіших особливостей її популярності була функція “напиши один раз – запускай де завгодно” (WORA). Ця функція дозволяє один раз написати програму Java та запустити її на будь-якій платформі. Наприклад, ви можете написати та компілювати програму Java на UNIX та запустити її Microsoft Windows, Macintosh або UNIX без будь-яких змін вихідного коду. WORA(Write once, run anywhere) досягається компіляцією програми Java на проміжну мову, яку називають байт-кодом. Формат байт-коду залежить від платформи. Віртуальна машина, яка називається Java Virtual Machine (JVM), використовується для запуску байтового коду на кожній платформі.

Зауважте, що JVM – це програма, реалізована в програмному забезпеченні. Це не фізична машина, і саме тому її називають “віртуальною” машиною. Завдання JVM – перетворити байт-код у виконуваний код відповідно до платформи, на якій він працює. Ця функція робить програми Java незалежно від платформи. Тобто та

сама програма Java може запускатися на декількох платформах без будь-яких модифікацій.

Нижче наведено кілька характеристик, які стоять за популярністю та прийняттям Java в галузі програмного забезпечення:

- простота;
- широка різноманітність середовищ використання;
- міцність.

Простота може бути суб'єктивним словом у цьому контексті. C ++ була популярною і потужною мовою програмування, яка широко використовувалася в індустрії програмного забезпечення на момент виходу Java. Якби ви були програмістом на C ++, Java забезпечила б вам простоту в навчанні та використанні над досвідом C ++. Java зберегла більшість синтаксису C / C ++, що було корисно для програмістів C / C ++, які намагалися вивчити цю нову мову. Ще краще, це виключило деякі з найбільш заплутані та важкі у використанні функції (хоча й потужні) C ++. Наприклад, у Java немає покажчиків та багаторазового успадкування, які присутні у C ++.

Якщо ви вивчаєте Java як свою першу мову програмування, то чи просто це вам вивчати, можливо, для вас не відповідає дійсності. Це причина, чому я сказав, що простота Java або будь-якої мови програмування є дуже суб'єктивною.

Мова Java та її бібліотеки (набір пакетів, що містять класи Java) зростали з моменту першого випуску. Потрібно буде докласти серйозних зусиль, щоб стати серйозним розробником Java.

Java може використовуватися для розробки програм, які можна використовувати в різних середовищах. Можете писати програми на Java, які можна використовувати в середовищі клієнт-сервер. Найпопулярнішим використанням програм Java в перші дні було розробка аплетів. Аплет – це програма Java, яка вбудована у веб-сторінку, яка використовує розмітку HyperText, мову (HTML) і відображається у веб-браузері, такому як Microsoft Internet Explorer, Google Chrome тощо. Код аплету зберігається на веб-сервері, завантажується на клієнтську машину, коли завантажується HTML-сторінка, що містить посилання на аплет за допомогою

браузера та запустіть на клієнтській машині. Java включає функції, які спрощують розробку розподілених додатків. Розподілений додаток складається з програм, що працюють на різних машинах, підключених через мережу. У Java є функції, які спрощують розробку одночасних програм.

Паралельна програма має кілька взаємодіючих потоків виконання, які працюють паралельно. Надійна програма стосується її здатності розумно вирішувати несподівані ситуації. Несподівана ситуація в програмі також відома як помилка. Java забезпечує надійність, надаючи безліч функцій для перевірки помилок у різних точках протягом життя програми.

Нижче наведено три різні типи помилок, які можуть виникнути в програмі Java:

- помилка часу компіляції;
- помилка під час виконання;
- логічна помилка.

Помилки часу компіляції також відомі як синтаксичні помилки. Вони викликані неправильним використанням синтаксису мови Java. Помилки часу компіляції виявляє компілятор Java. Програма з помилкою компіляції не збирається в байт-код, поки помилки не будуть виправлені. Пропущена крапку з комою в кінці виписки, призначивши десяткове значення, наприклад, 10.23 до змінної цілого типу тощо, є прикладами помилок часу компіляції.

Помилки під час виконання програми виникають під час запуску програми Java. Компілятор не виявляє такого роду помилки, оскільки компілятор не має всієї інформації про виконання, доступної йому. Java є сильно набраною мовою і має надійну перевірку типів під час компіляції, а також під час виконання. Java забезпечує чіткий механізм обробки винятків для оброблення помилки виконання. Коли в програмі Java виникає помилка виконання, JVM(Java Virtual Machine) видає виняток, який програма може наздогнати та вирішити. Наприклад, ділення цілого числа на нуль (наприклад, 17/0) генерує помилку виконання. Java уникає критичних помилок виконання, таких як перевищення пам’яті та витоку пам’яті, забезпечуючи вбудований механізм автоматичного розподілу пам’яті.

Логічні помилки – це найважливіші помилки програми, і їх важко знайти. Вони вводяться програмістом, неправильно реалізуючи функціональну вимогу. Таку помилку не може виявити компілятор Java або Java-виконання. Вони виявляються тестерами програм або користувачами, коли вони порівнюють фактичну поведінку програми з її очікуваною поведінкою. Іноді кілька логічних помилок можуть проникнути у виробниче середовище, і вони залишаються непоміченими навіть після виходу із програми.

Процес пошуку та виправлення помилок у програмі відомий як налагодження. Всі сучасні інтегровані середовища розробки (IDE), такі як NetBeans, Eclipse, JDeveloper, JBuilder тощо, надають програмістам інструмент, який називається налагоджувачем, який дозволяє їм запускати програму поетапно та перевіряти стан програми на кожному кроці для виявлення помилок. Налагодження – реальність щоденної діяльності програміста [14].

3.3. Середовище розробки Android Studio

Для того, щоб написати додаток для Android, нам знадобиться середовище розробки. Google зробив дуже корисний інструмент для всіх розробників Android, Android Studio. Android Studio – це офіційний IDE для розробки Android, і за один завантаження входить все необхідне для початку розробки програм для Android. До комплекту завантаження входить комплект розробки програмного забезпечення (SDK) з усіма бібліотеками Android, які нам можуть знадобитися, та інфраструктура для завантаження багатьох екземплярів Android-емулятора, щоб ми могли спочатку запустити наше додаток, не потребуючи реального пристрою [15].

Android Studio – нова і повністю інтегрована середовище розробки додатків, випущена компанією Google для операційної системи Android. Даний продукт покликаний забезпечити розробників новими інструментами для створення додатків.

При створенні нового проекту в Android Studio, буде показана структура проекту з усіма файлами, що містяться в каталозі.

Android Studio дозволяє вам побачити будь-які візуальні зміни, які ви робите в реальному часі в додатку. Ви також можете побачити, як ваш додаток буде одночасно виглядати на різних пристроях під управлінням Android, з різними настройками і дозволом екрану.

Продукт також володіє новими інструментами для упаковки і маркування коду. Це дозволить вам не загубитися в проекті, коли ви маєте справу з великою кількістю коду. У програмі також задіяна функція перетягування, завдяки якій можна переміщати компоненти за допомогою призначеного для користувача інтерфейсу.
Що ще пропонує Android Studio?

- надійне і просте середовище розробки;
- легко перевірити продуктивність програми на різних типах пристроїв;
- помічники і шаблони для загальних елементів програмування для Android;
- повнофункціональний редактор з безліччю додаткових інструментів, що сприяють прискоренню розробки додатків.

3.4. Інструмент автоматизації Selenium

Selenium – це програма тестування програмного забезпечення для веб-додатків, яка забезпечує уніфікований інтерфейс і працює майже з усіма браузерами, а тести можуть бути складені багатьма мовами. Selenium складається з декількох компонентів, таких як Selenium IDE, Selenium WebDriver, API клієнта Selenium, Сітка селену та Selenium RC (Selenium RC нещодавно застарів на користь Selenium WebDriver). Selenium IDE – це розширення Firefox, яке дозволяє тестеру записувати, редагувати та налагоджувати тести. Ядро Selenium включено до ID ID Selenium, що дозволяє легко та швидко зберігати тест у реальному середовищі під час роботи. Selenium – це програмне забезпечення з відкритим кодом, випущене під ліцензією

Apache 2.0. Це програмне забезпечення працює на різних платформах, таких як Linux / UNIX, Windows та OS X. Він вважається одним з перших популярних проектів з відкритим кодом, який спростив врешті-решт тестування на основі браузера для тестерів програмного забезпечення та розробників, а будь-які нещодавно випущені браузери можуть бути додані для підтримки швидко і легко, оскільки вони написані в чистому JavaScript. Оскільки двигун Selenium написаний на JavaScript, він викликає значну слабкість, оскільки браузери накладають дуже суворі моделі безпеки для будь-якого JavaScript, які виконуються браузерами для захисту користувачів від шкідливих сценаріїв. Наприклад, в Internet Explorer (IE) важче завантажувати файл, оскільки IE заважає JavaScript змінювати значення будь-якого елемента та переходити між доменами через єдину проблему політики щодо джерела хоста у IE.

Основне завдання Selenium Grid – запускати тести поруч на різних машинах проти різних браузерів або різних версій браузерів та операційних систем за допомогою WebDriver. Сітка селену дозволяє тестувальнику виконувати розподілене виконання тесту, до якого час може бути скорочений до повного набору тестів.

WebDriver, також відомий як “Selenium WebDriver” – це інструмент автоматизації тестування браузера для веб-додатків, який використовується для компіляції кінцевих тестів. WebDriver, який розроблений дуже простим способом забезпечити зручний інтерфейс API, щоб зробити його простішим у використанні, є більш стислими інтерфейсами програмування. Цей інструмент робить саме те, що очікував би кінцевий користувач за допомогою браузера: управління браузером автоматизовано, щоб кінцевий користувач повторював автоматизовані завдання. Це здається такою простою проблемою, яку потрібно вирішити, але за кадром необхідно виконати кілька завдань, перш ніж змусити її працювати.

Selenium WebDriver включив прив’язки мови та реалізацію індивідуального коду контролю браузера. WebDriver – лише одна складова Selenium серед багатьох. WebDriver не потребує підключення до віддаленого сервера для виконання свого завдання на рідній машині.

Реалізації WebDriver зазвичай відрізняються в кожному браузері та керують браузером з метою тестування на рідній машині із певним браузером і імітують взаємодію людини з веб-додатками.

З архітектури WebDriver відомо, що WebDriver безпосередньо викликає браузер, коли тести виконуються на рідній машині, використовуючи вбудовану підтримку браузерів для автоматизації на відміну від Selenium RC. Слід зазначити, що в Selenium RC архітектура працює по-іншому, коли вона передає HTTP-запити серверу, а сервер передає його до ядра Selenium. І навпаки, WebDriver не має проксі-сервера між клієнтом і браузером. Він безпосередньо передає селенські команди до ядра Selenium. Отже, API WebDriver - це більш потужний інструмент для прискорення часу виконання тестів порівняно з Selenium RC. Тести на основі мобільних браузерів раніше взагалі були неможливі за допомогою Selenium RC, але WebDriver може взаємодіяти з багатим вмістом API та мобільними програмами браузера. WebDriver підтримує майже всі найновіші версії браузерів там, на ринку. Офіційно зазначено, що всі майбутні вдосконалення можна робити лише в WebDriver.

І навпаки, що стосується веб-драйвера, він запускає браузер поза межами браузера, використовуючи API доступності [21].

3.5. Інструмент автоматизації Appium

Appium – це інструмент тестування з багатоплатформою автоматизації тестування з відкритим кодом. Він використовується для автоматизації тестових випадків для нативних, гібридних та веб-додатків. Інструмент має основну увагу як на додатках для Android, так і на iOS, і обмежився лише тестуванням домену мобільних додатків. Нещодавно, кілька оновлень назад, Appium також оголосив, що підтримуватиме тестування настільних додатків для Windows. Appium розширює функціональність WebDriver, щоб зробити важливі дії на мобільних пристроях, такі як можливості мультиконтактної роботи або можливі натискання фізичних кнопок.

Appium можна легко інтегрувати в існуючу сітку Selenium або використовувати як автономний каркас. Appium – це простий сервер HTTP, який записується за допомогою Javascript. Це відповідає загальній архітектурі клієнт-сервер. Сервер Appium обробляє запити від клієнта Appium та пересилає їх на тренажер / емулятор / реальний пристрій, де автоматизовані тестові сценарії (рисунок 3).

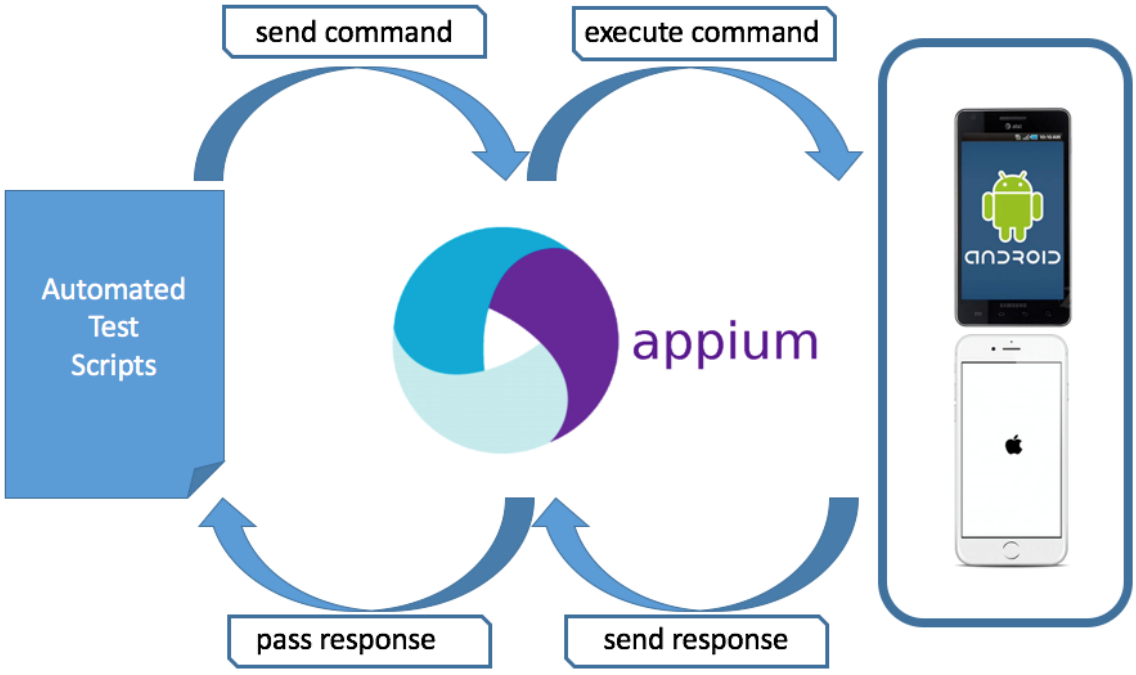

Рисунок 3 – Процес виконання тесту

Результати тесту повідомляються клієнту Appium через сервер Appium у вигляді відповіді сервера.

3.6. Засіб автоматизації роботи з програмними проектами Apache Maven

Maven – це інструмент управління та розуміння проектів, який надає розробникам повноцінну структуру життєвого циклу побудови. Команда розробників майже автоматично може автоматизувати інфраструктуру збирання проекту, оскільки

Maven використовує стандартний макет каталогів та життєвий цикл побудови за замовчуванням.

У випадку оточення декількох команд розробників Maven може створити спосіб роботи відповідно до стандартів за дуже короткий час. Оскільки більшість налаштувань проекту прості та багаторазові, Maven робить життя розробника легким під час створення звітів, перевірок, побудови та тестування налаштувань автоматизації.

Maven спрощує та стандартизує процес збирання проекту. Він без проблем обробляє компіляцію, розповсюдження, документацію, спільну роботу в команді та інші завдання. Maven збільшує багаторазове використання та піклується про більшість завдань, пов’язаних із побудовою.

3.7. Висновки до розділу

В даному розділі було розглянуто інформацію про засоби розробки. Проведено огляд середовища розробки IntelliJ IDEA 2019 і Android Studio та інструментів автоматизації Selenium і Appium.

Ознайомлення з мовою програмування Java, яка використовується для проведення тестування та представлення засобу автоматизації роботи з програмними проектами Apache Maven.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Важливою складовою виконання будь-якої роботи є розробка алгоритму та обміркований поетапний процес реалізації. Тож в цьому розділі описуються необхідні інструменти, компоненти, залежності для здійснення тестування.

4.1. Тестова документація

Перед реалізацією тестів слід враховувати всі нюанси будь-якого додатку та необхідні компоненти. Тому завжди тестування починається з написання тестової документації. Частина такої документації представлена нижче та в таблиці 4.1 і таблиці 4.2.

1. Ідентифікатор тесту: OP_01.

   Пріоритет: Med.

   Назва модуля: Екран входу в додаток.

   Назва тесту: Перевірка функціональності при запуску додатку.

   Опис: Підтвердити присутність модулів під час запуску початкової сторінки.

Таблиця 4.1 – Таблиця перевірки функціональності початкової сторінки

№	Тестові кроки	Тестові дані	Очікуваний результат	Фактичний результат	Статус
1	Відкриття початкової сторінки додатку		Початкова сторінка відкрилась	Початкова сторінка відкрилась	Проходить
2	Перевірка завантаження початкового зображення		Зображення присутнє	Зображення присутнє	Проходить

2. Ідентифікатор тесту: SU_02.

   Пріоритет: Med.

   Назва модуля: Реєстрація.

   Назва тесту: Перевірка функціональності реєстрації користувача без діабету.

   Опис: Підтвердити вірне виконання модулів реєстрації дійсних даних пацієнта, з відсутнім ступенем діабету.

Таблиця 4.2 – Таблиця перевірки функціональності реєстрації користувача

№	Тестові кроки	Тестові дані	Очікуваний результат	Фактичний результат	Статус
1	Натискання посилання реєстрації		Користувач перейшов на етап вибору ролі.	Користувач перейшов на етап вибору ролі.	Проходить
2	Перевірка відображення ролі користувача		Напис присутній	Напис присутній	Проходить
3	Натискання на роль “User”		Роль пацієнта обрано	Роль пацієнта обрано	Проходить
4	Натискання кнопки “Continue”		Перехід на сторінку обирання хвороби	Як і очікувалось	Проходить
5	Перевірка відображення напису про хвороби		Напис присутній	Як і очікувалось	Проходить
6	Перевірка статусу “без діабету”		Статус перевірений	Як і очікувалось	Проходить
7	Натискання кнопки “Continue”		Перехід на сторінку введення даних	Як і очікувалось	Проходить
8	Введення дійсного e-mail	forTest1@gmail.com	Дані можуть бути введені	Як і очікувалось	Проходить

Продовження таблиці 4.2

9	Введення дійсного Name	Simon	Дані можуть бути введені	Як і очікувалось	Проходить
10	Введення дійсного Surname	Killer	Дані можуть бути введені	Як і очікувалось	Проходить
11	Введення дійсного Password	Killer1987	Дані можуть бути введені	Як і очікувалось	Проходить
12	Введення Repeat Password	Killer1987	Дані можуть бути введені	Як і очікувалось	Проходить
13	Ввімкнення прапорця погодження з умовами і політикою конфіденційності		Зміна вказаного прапорця	Як і очікувалось	Проходить
14	Натискання кнопки “Sign up”		Поява спливаючого вікна	Поява спливаючого вікна	Проходить
15	Перевірка відображення повідомлення про успішну реєстрацію		Повідомлення відображене у вікні	Повідомлення відображене у вікні	Проходить
16	Натискання кнопки “Ok”		Відображення функціоналу початкової сторінки	Відображення функціоналу початкової сторінки	Проходить

Після написання тестової документації та продумування алгоритму створення-розпочинаємо процес реалізації системи.

4.2. Процес реалізації системи

Для процесу реалізації системи тестування, перш за все. треба встановити всі необхідні програми та середовища, додати компонент та описати можливості. Кожний обирає інструменти, які йому до вподоби, але було обрано ті, що зазначені в минулому розділі. Після створення проекту було підключено за допомогою Apache Maven необхідні компоненти для виконання зазначених робіт та полегшення процесу встановлення та використання. Частина підключених залежностей зображена на рисунку 4.1.

```
<dependencies>

    <dependency>
        <groupId>io.appium</groupId>
        <artifactId>java-client</artifactId>
        <version>7.3.0</version>
    </dependency>

    <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-server</artifactId>
        <version>3.141.59</version>
    </dependency>

    <dependency>
        <groupId>org.testng</groupId>
        <artifactId>testng</artifactId>
        <version>7.1.0</version>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>com.github.javafaker</groupId>
        <artifactId>javafaker</artifactId>
        <version>1.0.2</version>
    </dependency>
```

Рисунок 4.1 – Підключені залежності до проекту

Деякі з них вже представлені в попередньому розділі, а про інші трохи інформації. TestNG – це тестовий фреймворк, він допомагає задовольнити багато потреб в тестуванні. TestNG широко використовується разом з Selenium. NG означає “Next Generation” (“Наступне покоління”). TestNG схожий на JUnit (бібліотека для тестування програмного забезпечення для мови Java), але він більш потужний, коли справа стосується управління потоком виконання програми. Архітектура фреймворка допомагає зробити тести більш структурованими і забезпечити кращі точки валідації. Для додавання нових даних та редагування вже існуючих використовувалася JavaFaker – це бібліотека, яку можна використовувати для генерування широкого масиву реальних даних від адрес до посилань на популярну культуру.

Перед початком роботи з інструментом автоматизації Appium треба додати потрібний номер хосту та порту та розпочати роботу. На рисунку 4.2 зображена сторінка після відкриття програми – вона є необхідною для виконання вищезазначених дій.

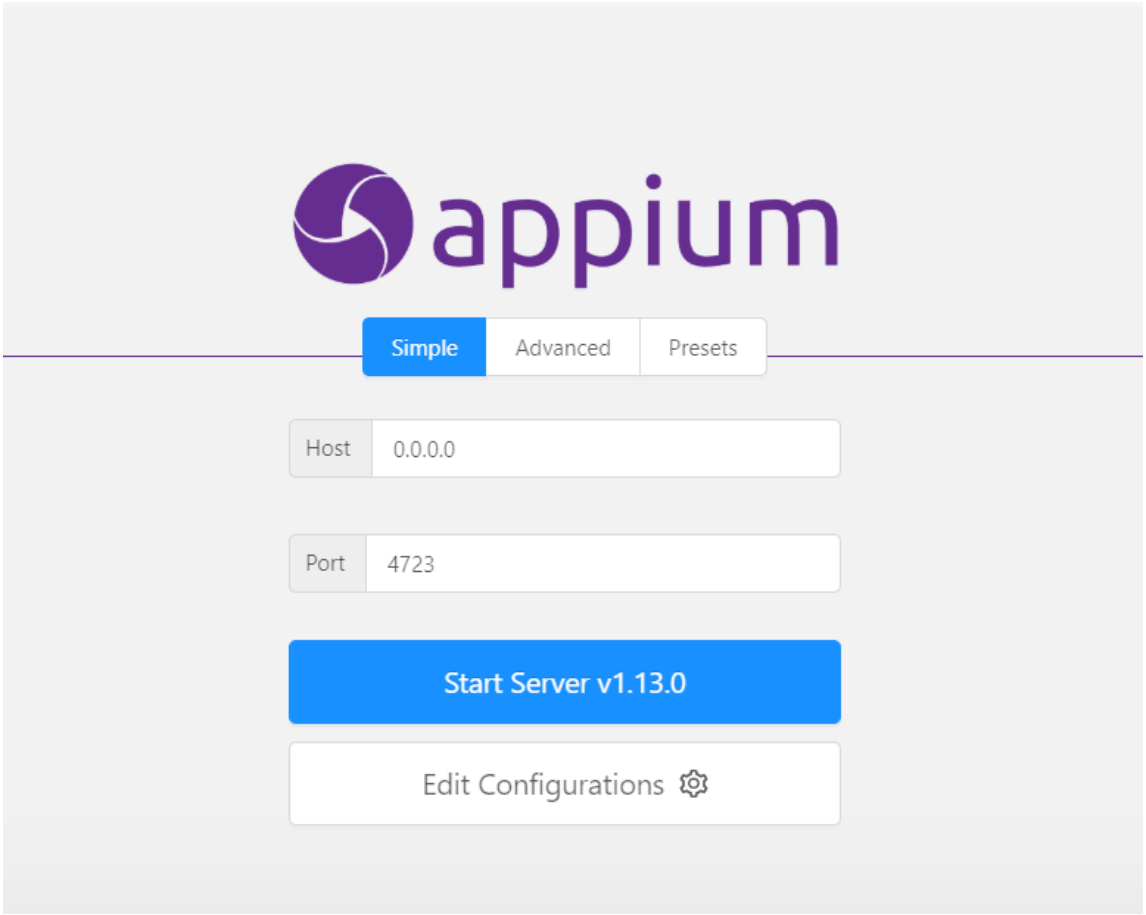

Рисунок 4.2 – Початкова сторінка інструменту автоматизації Appium

Після запуску сервера потрібно забезпечити створення сесії. Клієнтська бібліотека запрошує сервер на створення сесії. Сервер повертає sessionId, який використовується в наступних командах для взаємодії з додатками, що тестуються. Під час створення сесії зазначити певні desired capabilities(бажані можливості), які описують особливості створюваної сесії. Опис використаних бажаних можливостей зазначений в таблиці 4.3.

Таблиця 4.3 – Опис бажаних можливостей

Можливість	Опис
deviceName	Встановлює тип пристрою або емулятора, наприклад iPhone Simulator, iPad Simulator, iPhone Retina 4-inch, Android Emulator, Moto x, Nexus 5 і так далі. Приклад: caps.setCapability ("deviceName", "Nexus 5"); Або з використанням бібліотеки Appium: caps.setCapability (MobileCapabilityType.DEVICE_NAME, "Nexus 5");
platformName	Вказує операційну систему на мобільному пристрої. Можна вибрати зі значень iOS, Android і FirefoxOS. Приклад: caps.setCapability ("platformName", "Android"); Або з використанням бібліотеки Appium: caps.setCapability (MobileCapabilityType.PLATFORM_NAME, "Android");
appPackage	Визначає, який Java-пакет повинен бути запущений. Наприклад: com.android.calculator2 або com.android.settings caps.setCapability ("appPackage", "com.android.calculator2"); Або з використанням бібліотеки Appium: caps.setCapability (MobileCapabilityType.APP_PACKAGE, "com.android.calculator2");
appActivity	Визначає, яку Activity необхідно запустити з зазначеного пакета. Наприклад: MainActivity, .Settings, com.android.calculator2.Calculator caps.setCapability ("appActivity", "com.android.calculator2.Calculator"); Або з використанням бібліотеки Appium: caps.setCapability (MobileCapabilityType.APP_ACTIVITY, "com.android.calculator2.Calculator");

Для перегляду необхідних компонентів мобільного додатку та виконання створених тестів створено емулятор мобільного пристрою (рисунок 4.3), за допомогою середовища розробки Android Studio, з заданими критеріями та встановлено apk додатку.

Рисунок 4.3 – Емулятор мобільного пристрою

Після початку сесії з’являється вікно для вибору компонентів з мобільного додатку необхідних для тестування, область з інформацією про них та інші користі для процесу функції – це все зображено на рисунку 4.4.

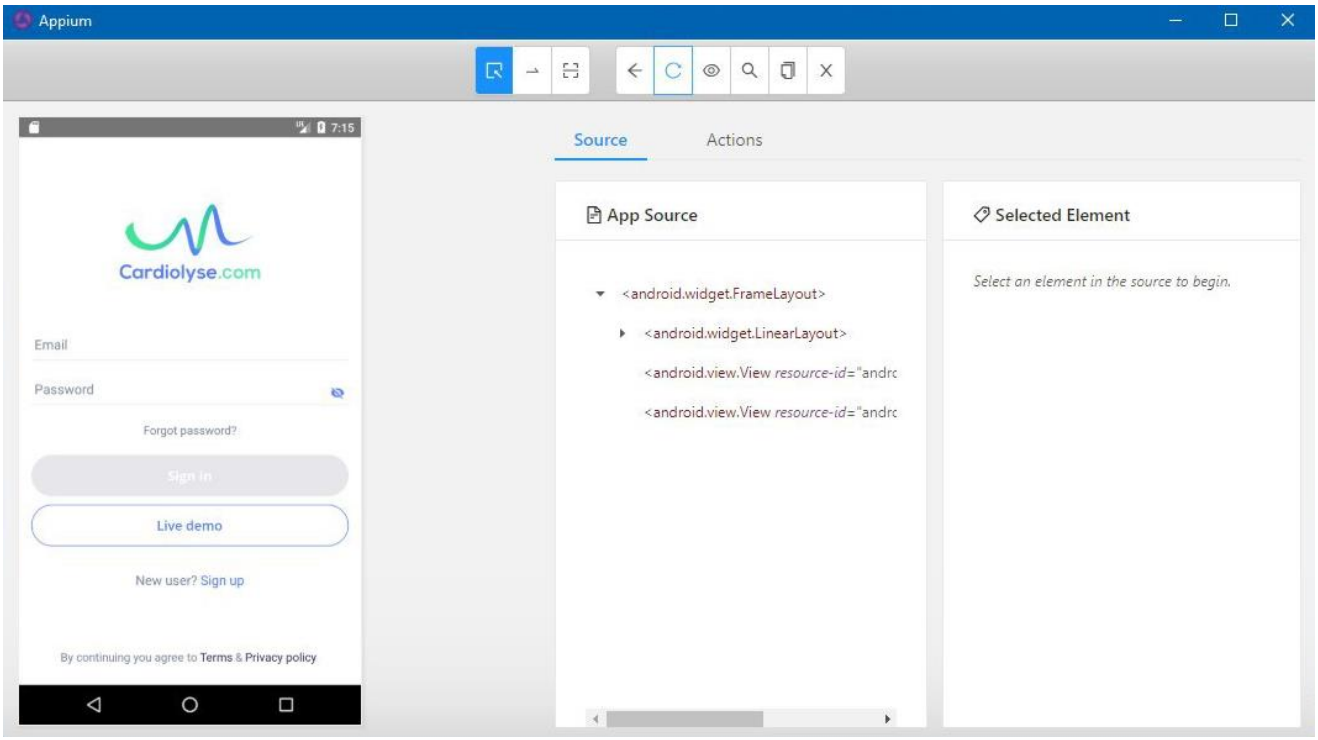

Рисунок 4.4 – Вікно сесії мобільного додатку

Після виконаних всіх підготовчих етапів розпочинається процес реалізації тестування. Першим кроком необхідно встановити прописані раніше бажані можливості, а далі користуючись вищезазначеними інструментами та розробленим алгоритмом виконувати написання тестів. Головним компонентом цієї дії є використання Page Object.

Page Object Model – це модель дизайну, яка стала популярною в автоматизації тестів для поліпшення обслуговування тесту та зменшення дублювання коду. Тести використовують методи цього класу об'єктів сторінки, коли їм потрібно взаємодіяти з інтерфейсом цієї сторінки, перевага полягає в тому, що якщо інтерфейс користувача зміниться, самі тести не потрібно змінювати, лише код у об’єкті сторінки потрібно змінити. Згодом усі зміни, які підтримують новий інтерфейс, розташовані в одному місці.

Збільшення покриття тестами для автоматизації може призвести до нездійсненної структури проекту, якщо локаторами не будуть керовані належним чином. Це може статися через дублювання коду або головним чином через дублювання використання локаторів. Оновлення локаторів елементів у дублюючому коді потребуватиме багато часу лише для коригування локаторів, тоді як цей час може бути витрачено на збільшення тестового покриття. Можна заощадити цей час, використовуючи Page Object Model у системі автоматизації тестування.

4.3. Висновки до розділу

В даному розділі було розглянуто частину тестової документації та інформацію про засоби реалізації системи. А саме про обрані компоненти, засоби, їх застосунок та демонстрацію під час початкових етапів підготовки, виконання та проведення тестування. Проведено огляд інструментів системи.

Ознайомлення з моделлю дизайну Page Object Model, яка надає поліпшення процесу тестування та зменшення дублювання коду.

5. РОБОТА З ПРОГРАМНОЮ СИСТЕМОЮ

В даному розділі представлені системні вимоги пристрою, необхідні для функціонування процесу тестування в повному обсязі, без певних виключень та інструкції для використання розробленої програмної системи.

5.1. Системні вимоги

Для забезпечення стабільної та коректної роботи розробленої системи тестування медичного додатку пристрій, на якому буде відбуватися весь процес, повинен мати декілька вимог:

- встановлену операційну систему 64-розрядну Microsoft Windows 10, Microsoft Windows 8, Mac OS або Linux;
- персональний комп’ютер повинен мати процесор не гірше Intel Core i3, або навіть краще Intel Core i7 з тактовою частотою вище 2 ГГц;
- з обсягом оперативної пам’яті не менше 6 Гб, а то і всі 16 Гб;
- відеокартою мінімум NVIDIA GeForce або ліпшою;
- на пристрої користувача має бути доступно 5 Гб вільного місця на диску, але рекомендується використовувати SSD диск.

Також користувачу необхідно мати доступ до інтернету зі середньою швидкістю 50 Мбіт/с.

5.2. Використання програмної системи

Для використання розробленої системи гарантування якості медичного додатку необхідно виконати поетапні пункти пошуку, встановлення, запуску та користування програмами, інструментами та компонентами. Тож для початку, якщо користувач не має встановленого середовища розробки IntelliJ IDEA, то він повинен зайти на офіційний сайт заданого продукту та встановити програму, звісно краще ж використовувати версію 2019-го року розробки або новішу. Після встановлення та розпакування відкрити необхідний проект в середовищі розробки. Виконати цей пункт можна за допомогою панелі інструментів, обравши вкладку File, що зображена на рисунку 5.1, та натиснувши на поле Open, після чого вписавши шлях до існуючого проекту та відкривши його.

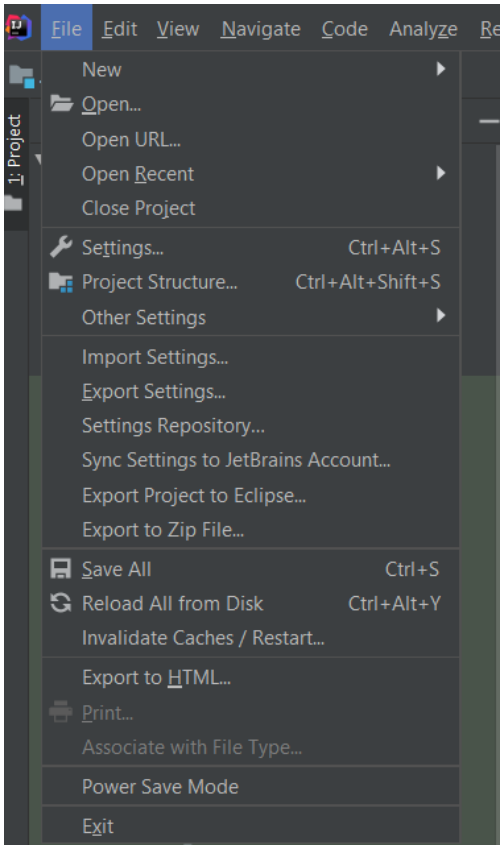

Рисунок 5.1 – Вкладка File на панелі інструментів середовища розробки IntelliJ IDEA

На наступному етапі відкрити вже існуючий або колись створений емулятор, для перегляду та перевірки тестів, якщо ж його немає то виконати декілька кроків. Зайти на офіційний сайт компанії розробників Android Studio та зберегти файл-установник на персональний комп'ютер, розпакувати та запустити середовище розробки. Після цього, для створення емулятору – пристрою для проведення тестування, на панелі інструментів обрати вкладку Tools (рисунок 5.2) та обрати поле AVD Manager. Далі натиснути кнопку Create Virtual Device і обрати всі необхідні критерії пристрою.

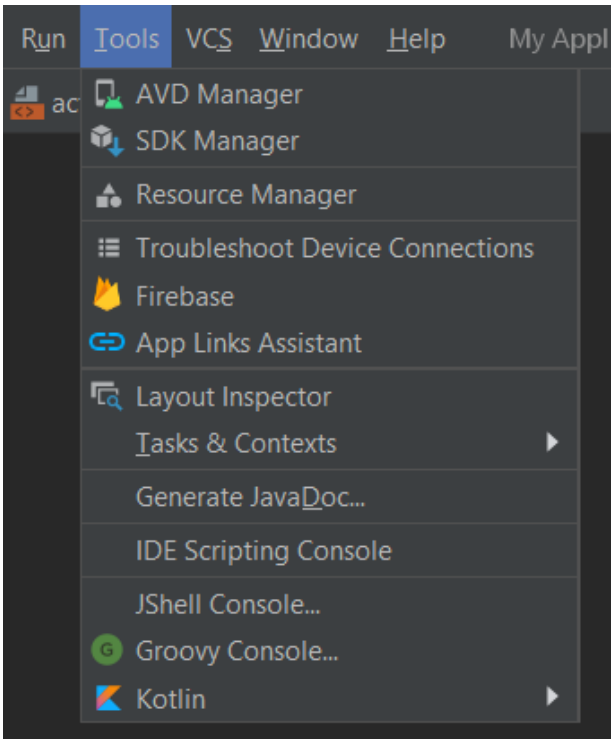

Рисунок 5.2 – Вкладка Tools на панелі інструментів
середовища розробки Android Studio

Запустити емулятор можливо через командний рядок або за допомогою Android Virtual Device Manager зображений на рисунку 5.3 та встановити apk медичного додатку.

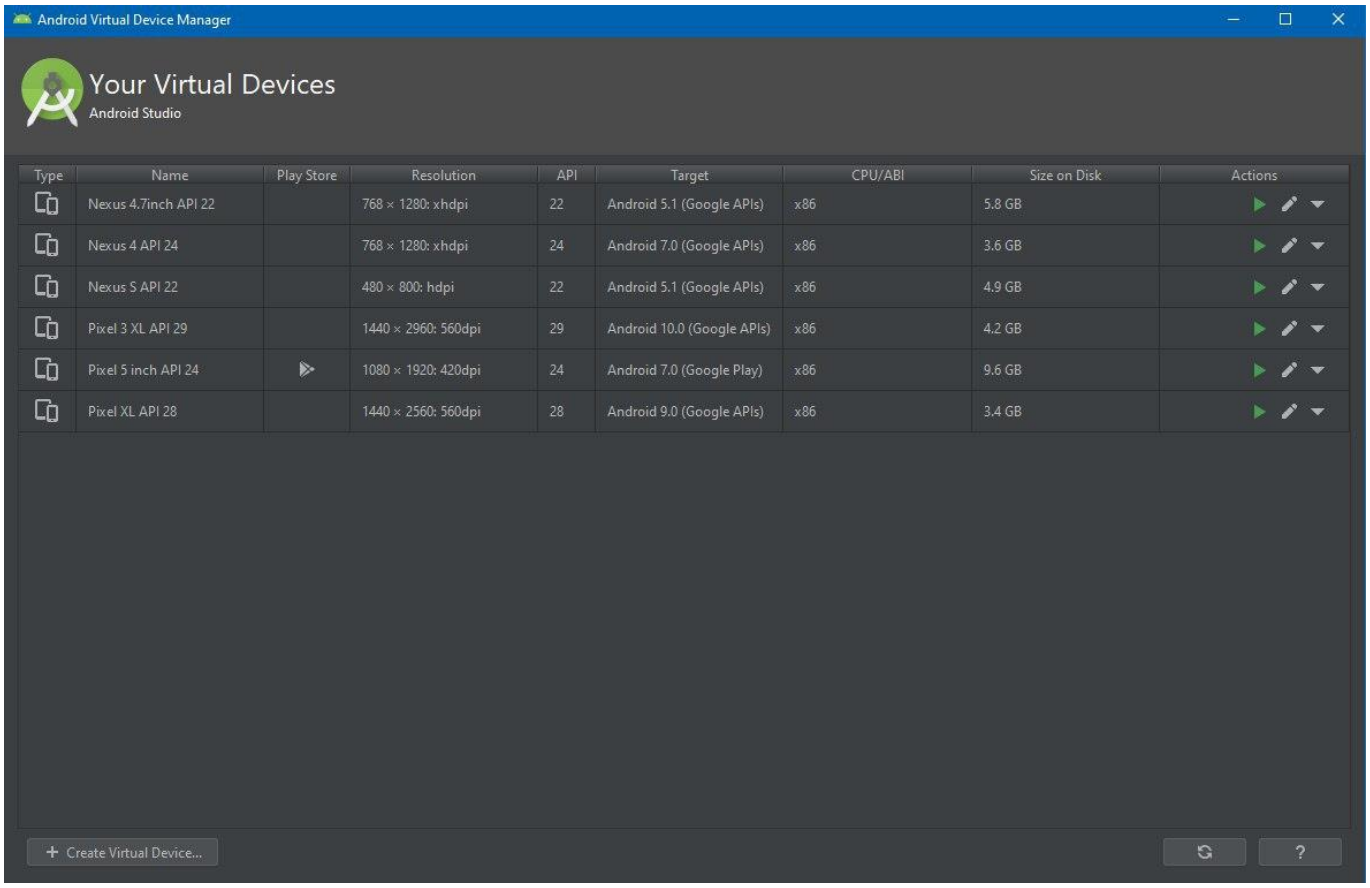

Рисунок 5.3 – Android Virtual Device Manager

Після цього відкрити або встановити інструмент автоматизації Appium та розпочати дію сервера. А далі запустити наш проект та почати процес тестування. Початковим тестом є перевірка відкриття центральної, вхідної сторінки медичного додатку. Зображення та виконання цього тесту показано на рисунку 5.4.

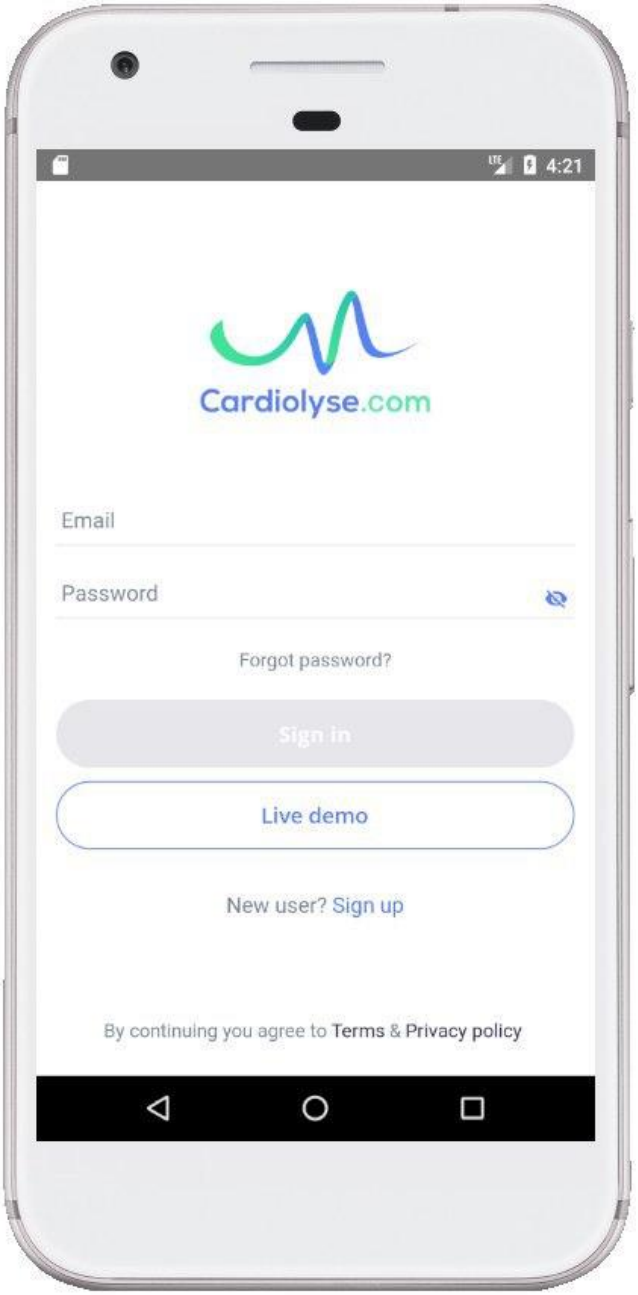

Рисунок 5.4 – Зображення початкового тесту

Процес виконання або не проходження тестів та час, за який кожен з них реалізується, демонструється користувачу відповідною відміткою (рисунок 5.5) поряд з проектом в IntelliJ IDEA.

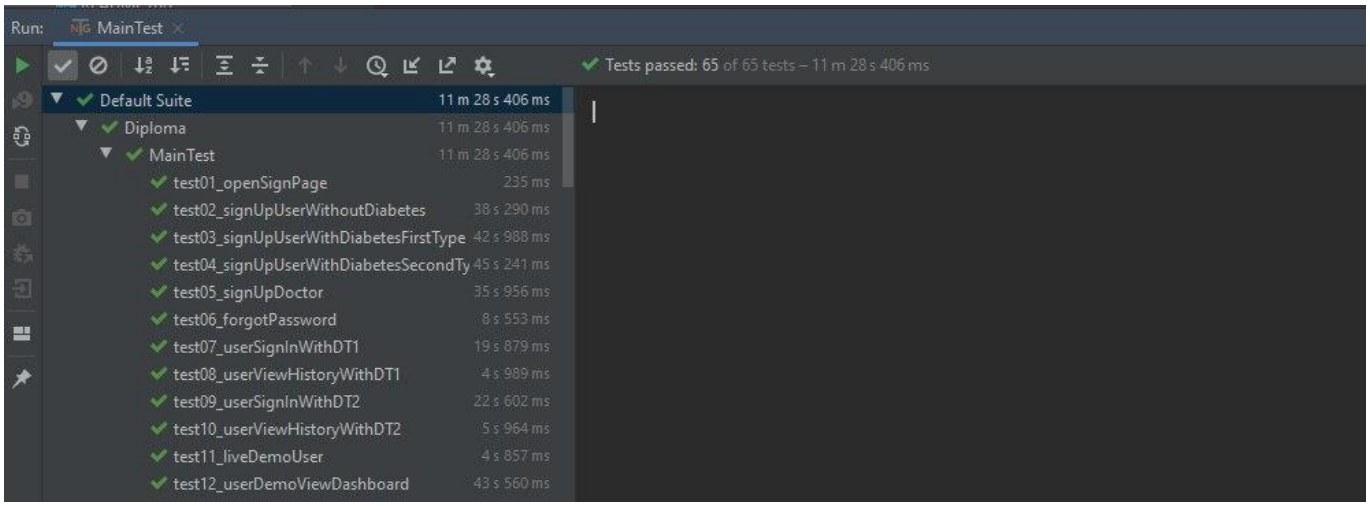

Рисунок 5.5 – Статус виконання тестів

За необхідності редагування виконаного проекту або додавання нових даних, для перегляду компонентів мобільного додатку потрібно розпочати створену сесію, про додавання, виконання та загальні дані якої містяться в попередньому розділі, яка містить головну інформацію про медичну систему.

5.3. Висновки до розділу

В даному розділі були розглянуті системні вимоги необхідні для використання розробленої системи. Також проведено покроковий огляд щодо користування процесом тестування та внесення певних змін за необхідністю.

ВИСНОВКИ

В ході виконання даної роботи було реалізовано систему гарантування якості мобільного, медичного додатку. Тестування необхідне для перевірки помилок, які допускають всі програмісти. Деякі з них можуть бути незначними, в той час як інші – мають руйнівні наслідки. Все, що виробляється людиною, може містити помилки. Саме тому будь-який продукт потребує перевірки – тестування, перш ніж його можна буде ефективно і безпечно використовувати.

Тестування має бути невід'ємною частиною розробки програмного забезпечення. На перший погляд може здатися, що метою тестування є показання, що програмне забезпечення повністю працює, але насправді це не зовсім так. Ціль гарантування якості – є виявлення певних недоліків, неполадок, одним словом показати, що система працює недосконало.

Існує багато факторів, які слід враховувати при виборі стратегії тестування. Основне правило для тестування: впровадити якомога більше автоматизованого тестування. Тестування вручну відбувається повільно і не вписується в сучасні спритні практики.

Тестування на мобільних пристроях абсолютно відрізняється від тестування на інших технологіях, таких як ноутбуки чи настільні комп'ютери. Найбільша різниця між мобільними та іншими технологіями полягає в тому, що користувач мобільного телефону знаходиться в русі, поки він використовує продукт. Тому дуже важливо знати про різні мережі передачі даних та різні типи мобільних пристроїв.

Для створення системи гарантування якості використовувалося середовище розробки IntelliJ IDEA, мову програмування для написання тестів Java та модель дизайну автоматизації тестів для поліпшення обслуговування тесту та зменшення дублювання коду Page Object Model. Інструментом автоматизації був обраний Appium, а приладом для проведення тестування емулятор, який був створений за допомогою середовища розробки Android Studio.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Куликов С. С. Тестирование программного обеспечения. Базовый курс. / С. С. Куликов. – Минск: ЧЕТЫРЕ ЧЕТВЕРТИ, 2017. – 312 с. – (2-е издание).
2. Myers G. The Art of Software Testing / G. Myers, C. Sandler, T. Badgett. – Canada: JohnWiley & Sons, 2012. – 240 с.
3. Copeland L. A Practitioner's Guide to Software Test Design / Lee Copeland. – London: Artech House, 2004. – 288 с.
4. Pettichord B. Lessons Learned in Software Testing / B. Pettichord, C. Kaner, J. Bach. – New York: Wiley, 2001. – 320 с. – (first edition).
5. Савин Р. Тестирование dot com. / Роман Савин. – Москва: ИЗДАТЕЛЬСТВО «ДЕЛО», 2007. – 312 с.
6. Knott D. Hands-On Mobile App Testing / Daniel Knott. – Crawfordsville: Addison-Wesley Professional, 2015. – 254 с.
7. Mobile testing tutorials point simply easy learning [Електронний ресурс] // Tutorials Point. – 2016. – Режим доступу до ресурсу: https://www.tutorialspoint.com/mobile_testing/index.htm.
8. Jacob J. Beginner's Guide for Mobile Applications Testing / Jeesmon Jacob., 2015. – 82 с.
9. DUANE K. IntelliJ IDEA in Action / DUANE K.. – United States of America: Manning Publications Co, 2006. – 450 с. – (1st edition).
10. Давыдов С. IntelliJ IDEA. профессиональное программирование на Java / С. Давыдов, А. Ефимов. – Санкт-Петербург: БХВ-Петербург, 2005. – 800 с.
11. Schildt H. Java: A Beginner's Guide / Herbert Schildt. – OSBORNE: 2018, 720. – (eight edition).
12. Sierra K. Head First Java (A Brain Friendly Guide) / K. Sierra, B. Bates., 2005. – 720 с. – (second edition).

13. Bloch J. Effective Java / Joshua Bloch. – United States: Addison-Wesley Professional, 2017. – 412 с. – (third edition).
14. Sharan K. Beginning Java 8 Fundamentals: Language Syntax, Arrays, Data Types, Objects, and Regular Expressions / Kishori Sharan., 2014. – 828 с. – (first edition).
15. Craig C. Learn Android Studio: Build Android Apps Quickly and Effectively / C. Craig, A. Gerber., 2015. – 486 с.
16. Smyth N. Android Studio Development Essentials - Android 7 Edition: Learn to Develop Android 7 Apps with Android Studio 2.2 / Neil Smyth., 2016. – 816 с. – (first edition).
17. Verma N. Mobile Test Automation with Appium: Mobile application testing made easy / Nishant Verma., 2017. – 256 с.
18. Hans M. Appium Essentials / Manoj Hans., 2015. – 188 с.
19. Garg N. Test Automation using Selenium WebDriver with Java: Step by Step Guide / Navneesh Garg., 2014. – 404 с. – (first edition).
20. Gundecha U. Learn Selenium: Build data-driven test frameworks for mobile and web applications with Selenium Web Driver 3 / U. Gundecha, C. Cocchiaro., 2019. – 536 с.
21. Rahman M. BROWSER-BASED AUTOMATION TESTING USING SELENIUM WEBDRIVER / Md.Foyzur Rahman., 2014. – 94 с.
22. Company S. Maven: The Definitive Guide / Sonatype Company. – Sebastopol: O'Reilly Media, 2008. – 470 с. – (first edition).
23. Siminiuc A. Improve Selenium Code with Automation Patterns: Page Object Model Page Factory Page Elements Base Page Loadable Component / Alex Siminiuc., 2017. – 112 с.

Додаток 1

Система гарантування якості медичних додатків в розподіленому середовищі

Специфікація

УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ ТМ62208_20Б

Аркушів 1

2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ ТМ62208_20Б 81-1	Записка.docx	Пояснювальна записка
Компоненти		
УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ ТМ62208_20Б 12-1	ConnectionSettings.java GroupsPage.java MainTest.java	Основні компоненти
УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ ТМ62208_20Б 13-1	Опис.docx	Опис програмних модулів
УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ ТМ62208_20Б 14-1	Тези.docx	Тези конференції

Додаток 2

Система гарантування якості медичних додатків в розподіленому середовищі

Текст програмного модулю

УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ ТМ62208_20Б 12-1

Аркушів 11

2020

```
package pageSettings;

import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;
import io.appium.java_client.remote.MobileCapabilityType;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;

import java.net.MalformedURLException;
import java.net.URL;

public class ConnectionSettings {
    public static AndroidDriver<AndroidElement> androidDriver;
    @BeforeClass
    public void connection() throws MalformedURLException, InterruptedException {

        DesiredCapabilities desiredCapabilities = new DesiredCapabilities();

        desiredCapabilities.setCapability(MobileCapabilityType.DEVICE_NAME,
"emulator-5554");
        desiredCapabilities.setCapability("platformName", "android");
        desiredCapabilities.setCapability("appPackage", "com.cardiolyse");
        desiredCapabilities.setCapability("appActivity", ".MainActivity");

        androidDriver = new AndroidDriver<AndroidElement>(new
URL("http://127.0.0.1:4723/wd/hub"), desiredCapabilities);
        Thread.sleep(3000, 30);
    }

    @AfterClass
    public void close(){

        androidDriver.quit();
    }

}


package pages;

import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;
import io.appium.java_client.pagefactory.AndroidFindBy;
import org.testng.Assert;

public class GroupsPage extends PatientsPage{

    @AndroidFindBy(xpath =
"/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.Fra
meLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.FrameL
ayout/android.view.ViewGroup/android.view.ViewGroup[2]/android.view.ViewGroup/android
.widget.Button[4]\n")
    private AndroidElement buttonGroups;

    @AndroidFindBy(xpath = "//android.view.ViewGroup[@content-
desc=\"groups_screen\"]/android.widget.ScrollView/android.view.ViewGroup/android.widg
et.TextView\n")
    private AndroidElement titleGroup;

    @AndroidFindBy(id = "group_T2DM")
    private AndroidElement buttonShowPatients;
```

```
    @AndroidFindBy(xpath = "//android.view.ViewGroup[@content-
desc=\"groups_screen\"]/android.widget.ScrollView/android.view.ViewGroup/android.view
.ViewGroup[5]/android.widget.TextView\n")
    private AndroidElement buttonCollapseGroups;

    @AndroidFindBy(id = "groups_search_field")
    private AndroidElement inputGroupName;

    @AndroidFindBy(xpath = "//android.view.ViewGroup[@content-
desc=\"groups_screen\"]/android.widget.ScrollView/android.view.ViewGroup/android.widg
et.ImageView\n")
    private AndroidElement iconNotFindGroup;

    @AndroidFindBy(xpath = "//android.view.ViewGroup[@content-
desc=\"groups_screen\"]/android.widget.ScrollView/android.view.ViewGroup/android.widg
et.TextView[2]")
    private AndroidElement titleNotFindGroup;

    @AndroidFindBy(id = "open_group_creating_button")
    private AndroidElement buttonCreateGroup;

    @AndroidFindBy(xpath = "//android.view.ViewGroup[@content-
desc=\"expand_all_groups_button\"]/android.widget.TextView")
    private AndroidElement buttonExpandAllGroup;

    @AndroidFindBy(xpath = "//android.view.ViewGroup[@content-
desc=\"groups_screen\"]/android.widget.ScrollView/android.view.ViewGroup/android.view
.ViewGroup[4]/android.widget.TextView\n")
    private AndroidElement buttonCollapseAllGroup;

    @AndroidFindBy(xpath = "//android.view.ViewGroup[@content-
desc=\"create_group_modal\"]/android.widget.TextView[1]\n")
    private AndroidElement titleNewGroup;

    @AndroidFindBy(xpath = "//android.view.ViewGroup[@content-
desc=\"create_group_modal\"]/android.widget.TextView[2]\n")
    private AndroidElement titleInstructionNewGroup;

    @AndroidFindBy(id = "create_group_field")
    private AndroidElement inputNewGroupName;

    @AndroidFindBy(id = "create_group_button")
    private AndroidElement buttonCreateNewGroup;

    @AndroidFindBy(xpath = "//android.view.ViewGroup[@content-
desc=\"groups_screen\"]/android.widget.ScrollView/android.view.ViewGroup/android.widg
et.TextView[2]\n")
    private AndroidElement titleAddFirstGroup;

    public GroupsPage(AndroidDriver<AndroidElement> androidDriver) {
        super(androidDriver);
    }

    public void clickButtonGroup(){
        buttonGroups.click();
    }

    public String getTextTitleGroup(){
        return titleGroup.getText();
    }

    public boolean titleGroup(){
        Assert.assertEquals(getTextTitleGroup(), "Groups");
        return true;
```

```
    }

    public void clickButtonShowPatients(){
        buttonShowPatients.click();
    }

    public void clickButtonCollapseGroups(){
        buttonCollapseGroups.click();
    }

    public void clickInputGroupName(){
        inputGroupName.click();
    }

    public void setValueGroupName(){
        inputGroupName.setValue("D1");
    }

    public boolean iconNotFindGroupIsDisplayed(){
        return iconNotFindGroup.isDisplayed();
    }

    public String getTextTitleNotFindGroup(){
        return titleNotFindGroup.getText();
    }

    public boolean titleNotFindGroup(){
        Assert.assertEquals(getTextTitleNotFindGroup(), "Did not find groups with
name");
        return true;
    }

    public void clickButtonCreateGroup(){
        buttonCreateGroup.click();
    }

    public void clickButtonExpandAllGroup(){
        buttonExpandAllGroup.click();
    }

    public void clickButtonCollapseAllGroup() {
        buttonCollapseAllGroup.click();
    }

    public String getTextTitleNewGroup(){
        return titleNewGroup.getText();
    }

    public boolean titleNewGroup(){
        Assert.assertEquals(getTextTitleNewGroup(), "Create new group");
        return true;
    }

    public String getTextTitleInstructionNewGroup(){
        return titleInstructionNewGroup.getText();
    }

    public boolean titleInstructionNewGroup(){
        Assert.assertEquals(getTextTitleInstructionNewGroup(), "Enter group name and
add patients in new group");
        return true;
    }

    public void clickInputNewGroupName() {
```

```
        inputNewGroupName.click();
    }

    public void setValueInputNewGroupName(String groupName) {
        inputNewGroupName.setValue(groupName);
    }

    public void clickButtonCreateNewGroup(){
        buttonCreateNewGroup.click();
    }

    public boolean buttonCreateGroupIsDisplayed(){
        return buttonCreateGroup.isDisplayed();
    }

    public String getTextTitleAddFirstGroup(){
        return titleAddFirstGroup.getText();
    }

    public boolean titleAddFirstGroup(){
        Assert.assertEquals(getTextTitleAddFirstGroup(), "to create your first
group");
        return true;
    }

    public boolean viewGroupDoctorDemoForCreatGroup() throws InterruptedException {
        clickButtonGroup();
        if(titleGroup()) {
            clickButtonCreateGroup();
            return true;
        }
        return false;
    }

    public boolean viewGroupDoctorDemo() throws InterruptedException {
        clickButtonGroup();
        if(titleGroup()){
            clickButtonShowPatients();
            sleepTime(2000);
            tap(500, 900);
            sleepTime(2000);
            clickComeBack();
            clickButtonCollapseGroups();
            clickInputGroupName();
            setValueGroupName();
            hideKeyboard();
            if (iconNotFindGroupIsDisplayed() && titleNotFindGroup()) {
                return true;
            }
        }
        return false;
    }

    public boolean viewGroupDoctor() throws InterruptedException {
        clickButtonGroup();
        if(titleGroup()){
            clickButtonExpandAllGroup();
            verticalSwipe(700, 1450, 400, 500);
            sleepTime(2000);
            verticalSwipe(700, 300, 1450, 500);
            sleepTime(2000);
            clickButtonCollapseAllGroup();
            return true;
        }
```

```
        return false;
    }

    public boolean createNewGroupDoctor() throws InterruptedException {
        clickButtonCreateGroup();
        sleepTime(2000);
        if(titleNewGroup() && titleInstructionNewGroup()){
            clickInputNewGroupName();
            setValueInputNewGroupName(newGroupName());
            hideKeyboard();
            sleepTime(2000);
            tap(1000, 600);
            tap(1000, 600);
            clickButtonCreateNewGroup();
            if(titleGroup()){
                return true;
            }
            return true;
        }
        return false;
    }

    public boolean viewGroupsWOPDoctor() throws InterruptedException {
        clickButtonGroup();
        sleepTime(5000);
        if(titleGroup() && buttonCreateGroupIsDisplayed()){
            return true;
        }
        return false;
    }

    public boolean createGroupsWOPDoctor() throws InterruptedException {
        clickButtonCreateGroup();
        if(titleNewGroup() && titleInstructionNewGroup()){
            clickComeBack();
            if(titleGroup()){
                return true;
            }
        }
        return false;
    }

}


package pageTests;

import org.testng.Assert;
import org.testng.annotations.Test;
import pageSettings.ConnectionSettings;
import pages.*;

public class MainTest extends ConnectionSettings {

    OpenPage openPage;
    RegistrationPage registrationPage;
    SignInPage signInPage;
    HistoryPageUser historyPageUser;
    SettingsPage settingsPage;
    LiveDemoPage liveDemoPage;
    GetStartedPage getStartedPage;
    MyDoctorPage myDoctorPage;
    HistoryPageDoctor historyPageDoctor;
    PatientsPage patientsPage;
```

```
    GroupsPage groupsPage;

    @Test
    public void test01_openSignPage() throws InterruptedException {
        openPage = new OpenPage(androidDriver);
        Assert.assertTrue(openPage.mainPageIsDisplayed());
    }

    @Test//(enabled = false, priority = 2)
    public void test02_signUpUserWithoutDiabetes() throws InterruptedException {
        registrationPage = new RegistrationPage(androidDriver);
        Assert.assertTrue(registrationPage.registrationUserWithoutDiabetes());
    }

    @Test
    public void test03_signUpUserWithDiabetesFirstType() throws InterruptedException
{
        Assert.assertTrue(registrationPage.registrationUserWithDiabetesFirstType());
    }

    @Test
    public void test04_signUpUserWithDiabetesSecondType() throws InterruptedException
{
        Assert.assertTrue(registrationPage.registrationUserWithDiabetesSecondType());
    }

    @Test
    public void test05_signUpDoctor() throws InterruptedException {
        Assert.assertTrue(registrationPage.registrationDoctor());
    }

    @Test
    public void test06_forgotPassword() throws InterruptedException {
        Assert.assertTrue(registrationPage.forgotPassword());
    }

    @Test
    public void test07_userSignInWithDT1() throws InterruptedException {
        signInPage = new SignInPage(androidDriver);
        Assert.assertTrue(signInPage.signInUserWithDiabetesFirstType());
    }

    @Test
    public void test08_userViewHistoryWithDT1() throws InterruptedException {
        historyPageUser = new HistoryPageUser(androidDriver);
        settingsPage = new SettingsPage(androidDriver);
        Assert.assertTrue(historyPageUser.viewHistoryUserWithDT1());
        Assert.assertTrue(settingsPage.signOut());
    }

    @Test
    public void test09_userSignInWithDT2() throws InterruptedException {
        Assert.assertTrue(signInPage.signInUserWithDiabetesSecondType());
    }

    @Test
    public void test10_userViewHistoryWithDT2() throws InterruptedException {
        Assert.assertTrue(historyPageUser.viewHistoryUserWithDT2());
        Assert.assertTrue(settingsPage.signOut());
    }

    @Test
    public void test11_liveDemoUser() throws InterruptedException {
        liveDemoPage = new LiveDemoPage(androidDriver);
```

```
        Assert.assertTrue(liveDemoPage.openDemoUser());
    }

    @Test
    public void test12_userDemoViewDashboard() throws InterruptedException {
        Assert.assertTrue(liveDemoPage.demoUserViewDashboard());
    }

    @Test
    public void test13_userDemoViewHistory() throws InterruptedException {
        Assert.assertTrue(historyPageUser.viewHistoryUserDemo());
    }

    @Test
    public void test14_userDemoHistoryViewDay() throws InterruptedException {
        Assert.assertTrue(historyPageUser.viewDayHistoryUserDemo());
    }

    @Test
    public void test15_userDemoGetStartedView() throws InterruptedException {
        getStartedPage = new GetStartedPage(androidDriver);
        Assert.assertTrue(getStartedPage.viewGetStartedUserDemo());
    }

    @Test
    public void test16_userDemoGetStartedSignUp() throws InterruptedException {
        Assert.assertTrue(getStartedPage.getStartedSignUpUserDemo());
    }

    @Test
    public void test17_userDemoGetStartedTap() throws InterruptedException {
        Assert.assertTrue(liveDemoPage.openDemoUser());
        Assert.assertTrue(getStartedPage.viewGetStartedUserDemo());
        Assert.assertTrue(getStartedPage.viewGetStartedTapUserDemo());
    }

    @Test
    public void test18_userDemoMyDoctorView() throws InterruptedException {
        myDoctorPage = new MyDoctorPage(androidDriver);
        Assert.assertTrue(myDoctorPage.viewMyDoctorUserDemo());
    }

    @Test
    public void test19_userDemoSettingsViewSignOut() throws InterruptedException {
        Assert.assertTrue(settingsPage.signOut());
    }

    @Test
    public void test20_liveDemoDoctor() throws InterruptedException {
        Assert.assertTrue(liveDemoPage.openDemoDoctor());
    }

    @Test
    public void test21_doctorDemoHistoryView() throws InterruptedException {
        historyPageDoctor = new HistoryPageDoctor(androidDriver);
        Assert.assertTrue(historyPageDoctor.viewHistoryDoctorDemo());
    }

    @Test
    public void test22_doctorDemoHistoryViewDayFirstPerson() throws
InterruptedException {
        Assert.assertTrue(historyPageDoctor.viewHistoryDayFirstPatientDoctorDemo());
    }
```

```
@Test
public void test23_doctorDemoHistoryViewDiagram() throws InterruptedException {
    Assert.assertTrue(historyPageDoctor.viewHistoryDiagramDoctorDemo());
}

@Test
public void test24_doctorDemoHistoryViewDay() throws InterruptedException {
    Assert.assertTrue(historyPageDoctor.viewHistoryDayDoctorDemo());
}

@Test
public void test25_doctorDemoHistoryViewDetails() throws InterruptedException {
    Assert.assertTrue(historyPageDoctor.viewHistoryDetailsDoctorDemo());
}

@Test
public void test26_doctorDemoViewPatients() throws InterruptedException {
    patientsPage = new PatientsPage(androidDriver);
    Assert.assertTrue(patientsPage.viewPatientsDoctorDemo());
}

@Test
public void test27_doctorDemoViewOnePatient() throws InterruptedException {
    Assert.assertTrue(patientsPage.viewOnePatientDoctorDemo());
}

@Test
public void test28_doctorDemoNewGroupView() throws InterruptedException {
    groupsPage = new GroupsPage(androidDriver);
    Assert.assertTrue(groupsPage.viewGroupDoctorDemoForCreatGroup());
    Assert.assertTrue(getStartedPage.viewNewGroupDoctorDemo());
}

@Test
public void test29_doctorDemoNewGroupSignUp() throws InterruptedException {
    Assert.assertTrue(getStartedPage.getStartedSignUpUserDemo());
}

@Test
public void test30_doctorDemoNewGroupTap() throws InterruptedException {
    Assert.assertTrue(liveDemoPage.openDemoDoctor());
    Assert.assertTrue(groupsPage.viewGroupDoctorDemoForCreatGroup());
    Assert.assertTrue(getStartedPage.viewNewGroupTapDoctorDemo());
}

@Test
public void test31_doctorDemoViewGroups() throws InterruptedException {
    Assert.assertTrue(groupsPage.viewGroupDoctorDemo());
}

@Test
public void test32_doctorDemoSettings() throws InterruptedException {
    Assert.assertTrue(settingsPage.settingsDoctorDemo());

}

@Test
public void test33_signInUser() throws InterruptedException {
    Assert.assertTrue(signInPage.signInUserWithout());
}

@Test
public void test34_userViewFirstCentralPage() throws InterruptedException {
    Assert.assertTrue(getStartedPage.viewFirstCentralPageUser());
```

```
}

@Test
public void test35_userViewDashboard() throws InterruptedException {
    Assert.assertTrue(historyPageUser.viewDashboardUser());
}

@Test
public void test36_userViewHistory() throws InterruptedException {
    Assert.assertTrue(historyPageUser.viewHistoryUser());
}

@Test
public void test37_userViewMyDoctor() throws InterruptedException {
    Assert.assertTrue(myDoctorPage.viewMyDoctorUserDemo());
}

@Test
public void test38_userViewSettings() throws InterruptedException {
    Assert.assertTrue(settingsPage.viewSettingsUser());
}

@Test
public void test39_userEditInformationPatient() throws InterruptedException {
    Assert.assertTrue(settingsPage.editInformationPatientUser());
}

@Test
public void test40_userChangeRole() throws InterruptedException {
    Assert.assertTrue(settingsPage.changeRolePatientUser());
}

@Test
public void test41_signOutUser() throws InterruptedException {
    Assert.assertTrue(settingsPage.signOut());
}

@Test
public void test42_signInDoctor() throws InterruptedException {
    Assert.assertTrue(signInPage.signInDoctorWithPatients());
}

@Test
public void test43_doctorViewHistory() throws InterruptedException {
    Assert.assertTrue(historyPageDoctor.viewHistoryDoctor());
}

@Test
public void test44_doctorViewPatients() throws InterruptedException {
    Assert.assertTrue(patientsPage.viewPatientsDoctor());
}

@Test
public void test45_doctorCreateNewPatient() throws InterruptedException {
    Assert.assertTrue(patientsPage.createNewPatientDoctor());
}


@Test
public void test46_doctorEditPatient() throws InterruptedException {
    Assert.assertTrue(patientsPage.editPatientDoctor());
}

@Test
```

```
    public void test47_doctorViewCentralPage() throws InterruptedException {
        Assert.assertTrue(getStartedPage.viewCentralPageDoctor());
    }

    @Test
    public void test48_doctorViewGroup() throws InterruptedException {
        Assert.assertTrue(groupsPage.viewGroupDoctor());
    }

    @Test
    public void test49_doctorCreateNewGroup() throws InterruptedException {
        Assert.assertTrue(groupsPage.createNewGroupDoctor());
    }

    @Test
    public void test50_signOutDoctor() throws InterruptedException {
        Assert.assertTrue(settingsPage.signOutDoctor());
    }

    @Test
    public void test51_signInDoctorWithoutPatients() throws InterruptedException {
        Assert.assertTrue(signInPage.signInDoctorWithoutPatients());
    }

    @Test
    public void test52_doctorViewHistoryWOP() throws InterruptedException {
        Assert.assertTrue(historyPageDoctor.viewHistoryDoctor());
    }

    @Test
    public void test53_doctorViewPatientsWOP() throws InterruptedException {
        Assert.assertTrue(patientsPage.viewPatientsWOPDoctor());
    }

    @Test
    public void test54_doctorPressAddPatientsWOP() throws InterruptedException {
        Assert.assertTrue(patientsPage.pressAddPatientWOPDoctor());
    }

    @Test
    public void test55_doctorViewCentralPageWOP() throws InterruptedException {
        Assert.assertTrue(getStartedPage.viewCentralPageWOPDoctor());
    }

    @Test
    public void test56_doctorViewCentralPageWOPWithTap() throws InterruptedException
{
        Assert.assertTrue(getStartedPage.viewCentralPageTapWOPDoctor());
    }

    @Test
    public void test57_doctorViewGroupsWOP() throws InterruptedException {
        Assert.assertTrue(groupsPage.viewGroupsWOPDoctor());
    }

    @Test
    public void test58_doctorCreateGroupsWOP() throws InterruptedException {
        Assert.assertTrue(groupsPage.createGroupsWOPDoctor());
    }

    @Test
    public void test59_doctorViewSettingsWOP() throws InterruptedException {
        Assert.assertTrue(settingsPage.signOutDoctor());
    }
```

```
    @Test
    public void test60_requiredEmailAndPasswordSignIn() throws InterruptedException {
        Assert.assertTrue(signInPage.signInRequiredEmailAndPassword());
    }

    @Test
    public void test61_invalidEmailSignIn() throws InterruptedException {
        Assert.assertTrue(signInPage.signInInvalidEmail());
    }

    @Test
    public void test62_shortPasswordSignIn() throws InterruptedException {
        Assert.assertTrue(signInPage.signInShortPassword());
    }

    @Test
    public void test63_incorrectPasswordOrEmailSignIn() throws InterruptedException {
        Assert.assertTrue(signInPage.signInIncorrectPasswordOrEmail());
    }

    @Test
    public void test64_invalidNameAndSurnameSignUp() throws InterruptedException {
        Assert.assertTrue(registrationPage.invalidNameAndSurnameRegistration());
    }

    @Test
    public void test65_dontMatchPasswordSignUp() throws InterruptedException {
        Assert.assertTrue(registrationPage.dontMatchPasswordRegistration());
    }

}
```

Додаток З

Система гарантування якості медичних додатків в розподіленому середовищі

Опис програмного модулю

УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ ТМ62208_20Б 13-1

Аркушів 7

2020

АНОТАЦІЯ

Додаток містить опис системи гарантування якості медичного додатку. В доданій частині системи зображені такі функції:

– налаштування та підключення пристрою;
– один з створених Page Object для тестування функціоналу вкладки групи;
– виклик та виконання тестів системи.

Вхідні та вихідні дані отримуються засобами реалізованими Java та інструментом автоматизації Appium.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

У цьому додатку описано систему гарантування якості медичного додатку. Модулі системи описані в додатку 2.

Система працює в операційних системах, таких як Windows8, Windows10.

Компоненти необхідні для установки: IntelliJ IDEA, Android Studio, Appium. Використана мова програмування – Java.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблена система призначена для гарантування якості медичного додатку.

Дана система може бути використана компанією для покращення функціоналу, загальних можливостей додатку та визначення недоліків, які необхідно усунути для подальшого розроблення або при оновленні існуючої системи.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Система являє собою програму зі зручним та зрозумілим використанням, який дозволяє швидко перевірити та проаналізувати справність медичного додатку.

Для роботи з системою потрібно встановити всі додаткові програмні додатки та запустити необхідні компоненти. Після цього можна працювати з розробленою системою, а за необхідності змінювати та додавати нові деталі, робити аналіз та опрацьовувати отримані результати.

ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Компоненти необхідні для використання системи гарантування якості: IntelliJ IDEA, Android Studio, Appium. Використана мова програмування для системи – Java.

Розроблена система працює в операційних системах Windows8, Windows10 та потребує використання додаткових компонентів: Apache Maven, Selenium, Page Object, TestNG, JavaFaker.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Для використання розробленої системи гарантування якості медичного додатку необхідно виконати поетапні пункти пошуку, встановлення, запуску та користування програмами, інструментами та компонентами. Відкрити необхідний проект в середовищі розробки. Запустити вже існуючий або колись створений емулятор, для перегляду та перевірки тестів. Далі відкрити інструмент автоматизації Appium та розпочати дію сервера. Після успішного виконання всіх етапів запустити наш проект та почати процес тестування.

Додаток 4

Система гарантування якості медичних додатків в розподіленому середовищі

Тези конференції

УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ ТМ62208_20Б 14-1

Аркушів 2

2020

Сучасна цивілізація досягла високого розвитку, але прогрес не стоїть на місці: покращуються та оновлюються вже існуючі технології, створюються нові, більш довершені та необхідні для полегшення та оптимізації умов життя людей.

Останнім часом більшість населення широко застосовує технологічні здобутки. До них належать і телефонні пристрої. Цей факт сприяє створенню великої кількості програмних продуктів, спрямованих для використання на операційних системах і платформах для мобільних телефонів та планшетних комп'ютерів Android та iOS. На сьогоднішній день масивний об’єм операцій, які раніше виконувались лише в певному закладі, людина має змогу обробити вдома, не витрачаючи на це багато часу. Це стосується різноманітних дій: починаючи з оплати товарів з інтернет-магазину і закінчуючи використанням медичних додатків для отримання направлення від лікаря на необхідну процедуру. У найближчому майбутньому людство планує перевести в режим мобільного користування велику кількість операцій, які є базовими та необхідними для комфортного проживання, розвитку та вдосконалення.

Зараз широкого поширення набувають медичні додатки.

Це пов’язано з тим що, вони зберігають час лікарів та пацієнтів; пацієнт має вільний доступ до перегляду результатів аналізів, історії хвороби та внесення необхідних даних, які стосуються стану здоров’я; більш точно можна визначити діагноз, його особливості та способи лікування. Тому додатки необхідно ретельно тестувати, щоб мати точні введені дані та результати аналізу і адекватну роботу програмного продукту.

Тестування програмного забезпечення – процес перевірки відповідності заявлених до продукту вимог і реально реалізованої функціональності, здійснюваний шляхом спостереження за його роботою в штучно створених ситуаціях і на обмеженому наборі тестів, обраних певним чином. Техніка тестування також включає як процес пошуку помилок або інших дефектів, так і випробування програмних частин з метою оцінки. Також тестування використовується перед оновленням продукту до більш оптимізованого виду. Тому для перевірки медичного додатку на відповідність заявленим функціям та характеристикам було написано автотести за допомогою

Appium – інструмента автоматизації з відкритим кодом для запуску скриптів та тестування нативних програм, мобільних веб-додатків та гібридних додатків на Android або iOS за допомогою веб-диска та використано об'єктно-орієнтовану мову програмування Java.

Таким чином, можна зробити висновок, що тестування є одним з найважливіших та ключових етапів розробки програмного продукту, особливо, якщо додаток відноситься до медичної сфери застосування і пов’язаний зі здоров’ям людини.

Перелік посилань:

1. ЩО ТАКЕ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ЯКЕ ЙОГО ЗНАЧЕННЯ? [Електронний ресурс] // QAinfo. – 2018. – Режим доступу до ресурсу: https://www.quality-assurance-group.com/shho-take-testuvannya-programnogo-zabezpechennya-ta-yake-jogo-znachennya/.
2. Manoj H. Appium Essentials / Hans Manoj., 2015. – 190 c. – (Packt Publishing).